
Microsoft SQL Server Always Encrypted: Integration Guide

THALES LUNA HSM

Document Information

Document Part Number	007-001504-001
Revision	A
Release Date	14 January 2022

Trademarks, Copyrights, and Third-Party Software

Copyright © 2021 Thales Group. All rights reserved. Thales and the Thales logo are trademarks and service marks of Thales Group and/or its subsidiaries and are registered in certain countries. All other trademarks and service marks, whether registered or not in specific countries, are the property of their respective owners.

CONTENTS

Overview	4
Third Party Application Details	4
SQL Server Always Encrypted Setup.....	5
Using Multiple Clients Application Servers	5
Certified Platforms	6
Prerequisites	6
Configuring Luna HSM	6
Integrating Luna HSM with SQL Server Always Encrypted	7
Configure SafeNet KSP	8
Create a Sample Database	10
Generate Column Master Key	12
Generate Column Encryption Key.....	16
Encrypt Columns using Always Encrypted Keys	18
View Always Encrypted Data	20
Rotate Always Encrypted Key.....	25
Remove Always Encrypted Column Encryption.....	32
Contacting Customer Support.....	37
Customer Support Portal	37
Telephone Support.....	37
Email Support.....	37

Overview

This document contains instructions for integrating Microsoft SQL Server Always Encrypted with Luna HSM devices. In case of Microsoft SQL Server Always Encrypted, the term “Always” implies that data is encrypted all the time; not just at rest, but also during flight. Furthermore, the encryption keys themselves – which are essential for both encrypting and decrypting – are not stored in the database. Those keys stay with you, at the client side. Data when it arrives at the client can be decrypted on the client and by the client, who possesses the necessary keys. Likewise, when inserting or updating new data, that data gets encrypted immediately on the client, before it ever leaves the client, and remains encrypted in-flight all the way to the database server, where SQL Server can only store it in that encrypted state – it cannot decrypt it.

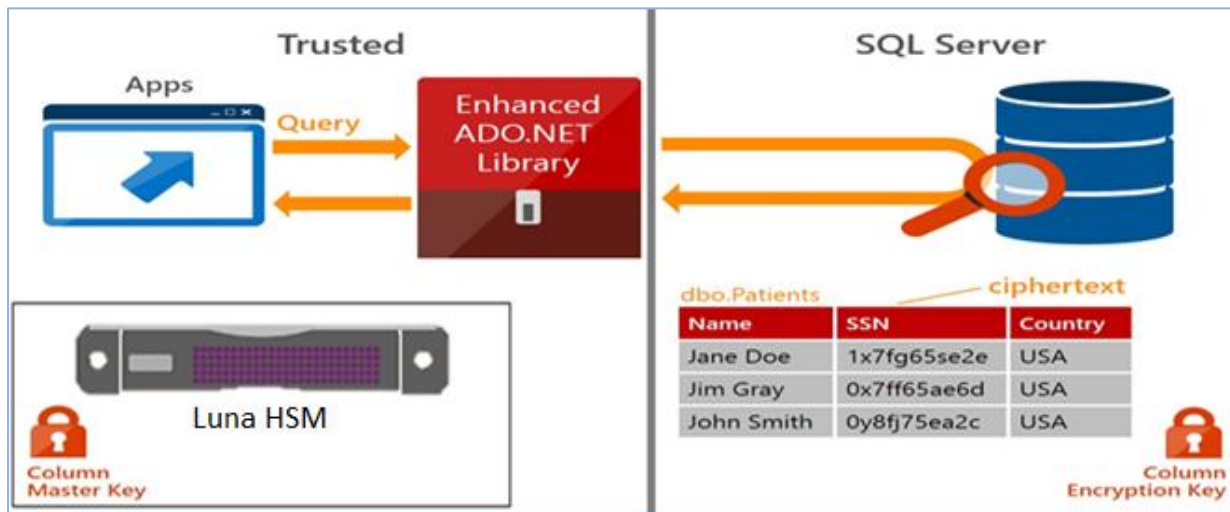
Always Encrypted is a hybrid feature, where all the encryption and decryption occurs exclusively on the client side. Using Luna HSMs with Microsoft SQL Server Always Encrypted provides the following benefits:

- > Secure generation, storage and protection of the encryption key on FIPS 140-2 level 3 validated hardware.
- > Full life cycle management of the keys.
- > HSM audit trail.
- > Significant performance improvements by off-loading cryptographic operations from application servers.

Third Party Application Details

This integration uses the following third party applications:

Microsoft SQL Server Always Encrypted



The new feature, called Always Encrypted, is available from SQL Server 2016’s first public preview. Always Encrypted adds an extra measure of security while the data is being used, when it is most susceptible to attack. The new security layer addresses that vulnerability by keeping the data encrypted during transactions and computations, and by only giving decryption keys to the client. If anyone else, including a database or system administrator, tries to access that client’s database, sensitive data will not appear in plaintext.

In order to use SQL Always Encrypted, the following keys are created:

- > Column master key
- > Column encryption key

A column encryption key is used to encrypt data in an encrypted column. A column master key is a key-protecting key that encrypts one or more column encryption keys. The Database Engine stores encryption configuration for each column in database metadata. However, the Database Engine never stores or uses the keys of either type in plaintext. It only stores encrypted values of column encryption keys and the information about the location of column master keys, which are stored in external trusted key stores, such as a Hardware Security Module (HSM).

For more information about SQL Server Always Encrypted, refer:

<https://docs.microsoft.com/en-us/sql/relational-databases/security/encryption/always-encrypted-database-engine?view=sql-server-ver15>

SQL Server Always Encrypted Setup

This integration uses the following setup to demonstrate the Always Encrypted with Luna HSM.

1. A Windows Server machine, which will become a Domain Controller.
2. A Windows Server machine, which will become SQL Server.
3. A Windows 10 Professional machine which will become the client.
4. A Windows 10 Professional machine which will become another client.

The machines utilized in the setup are as follows:

DC: Windows Server 2019 Standard Edition as Domain Controller.

DB: Windows Server 2019 Standard Edition as SQL Server 2019 Enterprise Edition.

CL1: Windows 10 Professional machine as Client1 for Provisioning Always Encrypted using Luna HSM.

CL2: Windows 10 Professional machine as Client2 for decrypting the encrypted data if and only if Column Master Key is accessible.

NOTE: This integration will use the same names as described above at various steps for demonstration.

Using Multiple Clients Application Servers

Each client server requiring access to the contents of data encrypted with a given CEK must have access to the CMK created and stored on Luna HSM partition. On each client, ensure that Luna Client is installed and KSP is registered using the same user who has generated the CMK on Luna HSM partition.

Certified Platforms

This integration is certified on the following platforms:

Luna HSM: Luna HSM appliances are purposefully designed to provide a balance of security, high performance, and usability that makes them an ideal choice for enterprise, financial, and government organizations. Luna HSMs physically and logically secure cryptographic keys and accelerate cryptographic processing. The Luna HSM on premise offerings include the Luna Network HSM, Luna PCIe HSM, and Luna USB HSMs. Luna HSMs are also available for access as an offering from cloud service providers such as IBM cloud HSM and AWS cloud HSM classic.

Platforms Tested	Luna HSM	Microsoft SQL Server
Windows Server 2019	Luna SA v7.7.0 Luna Client v10.4.0	Microsoft SQL Server 2019

NOTE: This integration is tested in both HA and FIPS mode.

Prerequisites

Before you proceed with the integration, complete the following tasks on each client:

Configuring Luna HSM

If you are using Luna HSM:

1. Verify the HSM is set up, initialized, provisioned and ready for deployment.
2. Create a partition that will be later used by SQL Server Always Encrypted Client machine.
3. If using a Luna Network HSM, register a client for the system and assign the client to the partition to create an NTLS connection. Initialize the Crypto Officer and Crypto User roles for the registered partition.
4. Ensure that the partition is successfully registered and configured. The command to see the registered partitions is:

```
C:\Program Files\SafeNet\LunaClient>lunacm.exe
```

```
lunacm.exe (64-bit) v10.4.0-417. Copyright (c) 2021 SafeNet. All rights reserved.
```

```
Available HSMs:
```

```
Slot Id -> 0
```

```
Label -> SQLAE
```

```
Serial Number -> 1312109861428
```

```
Model -> LunaSA 7.7.0
```

```
Firmware Version -> 7.7.0
```

```
Bootloader Version -> 1.1.2
```

```
Configuration -> Luna User Partition With SO (PW) Key Export With
```

	Cloning Mode
Slot Description ->	Net Token Slot
FM HW Status ->	Non-FM

- For PED-authenticated HSM, enable partition policies 22 and 23 to allow activation and auto-activation.

NOTE: Refer to [Luna HSM documentation](#) for detailed steps on creating NTLS connection, initializing the partitions, and assigning various user roles.

Set up Luna HSM High-Availability

Refer to the [Luna HSM documentation](#) for HA steps and details regarding configuring and setting up two or more HSM boxes on host systems. You must enable the HAOnly setting in HA for failover to work so that if the primary goes down due to any reason all calls automatically route to the secondary until the primary recovers and starts up.

Set up Luna HSM in FIPS Mode

NOTE: This setting is not required for Universal Client. This setting is applicable only for Luna Client 7.x.

Under FIPS 186-3/4, the RSA methods permitted for generating keys are 186-3 with primes and 186-3 with aux primes. This means that RSA PKCS and X9.31 key generation is no longer approved for operation in a FIPS-compliant HSM. If you are using the Luna HSM in FIPS mode, you have to make the following change in the configuration file:

```
[Misc]
RSAKeyGenMechRemap=1
```

The above setting redirects the older calling mechanism to a new approved mechanism when Luna HSM is in FIPS mode.

Integrating Luna HSM with SQL Server Always Encrypted

This section contains detailed instructions to integrate Microsoft SQL Server Always Encrypted with a Luna HSM. Following are the processes involved in provisioning and configuring the Always Encrypted using Luna HSM.

- > [Configure SafeNet KSP](#)
- > [Create a Sample Database](#)
- > [Generate Column Master Key](#)
- > [Generate Column Encryption Key](#)
- > [Encrypt Columns using Always Encrypted Keys](#)
- > [View Always Encrypted Data](#)
- > [Rotate Always Encrypted Key](#)
- > [Remove Always Encrypted Column Encryption](#)

Configure SafeNet KSP

Register the SafeNet Key Storage Provider (KSP) on the target client machine. In case of multiple client servers, SafeNet KSP needs to be register on each and every client server requiring access to always encrypted columns in a database.

To configure the SafeNet KSP on **CL1**, login as server administrator and perform the following steps:

1. Navigate to the **SafeNet KSP** directory.

```
<Luna Client Installation Directory>\KSP
```

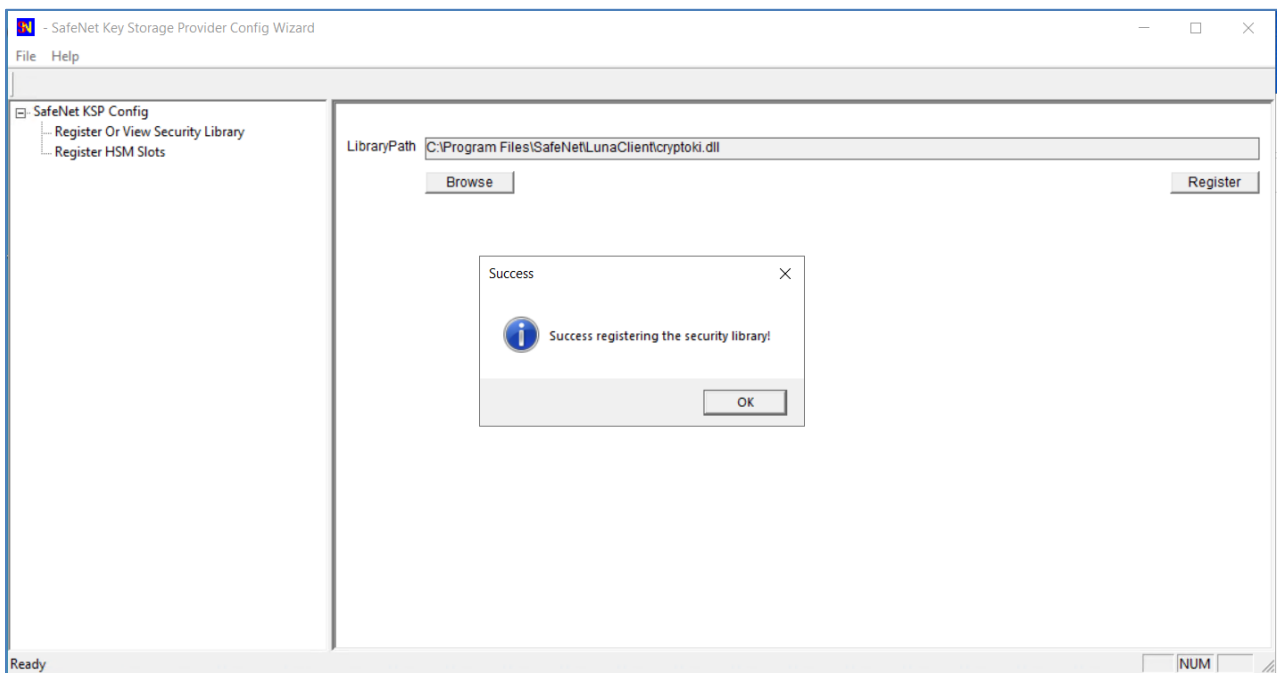
For Example: `cd "C:\Program Files\SafeNet\LunaClient\KSP"`

2. Run the **KSPConfig.exe** (KSP configuration wizard) utility to register the SafeNet KSP through a GUI. The general form of command is:

```
<Luna Client Installation Directory>\KSP>KspConfig.exe
```

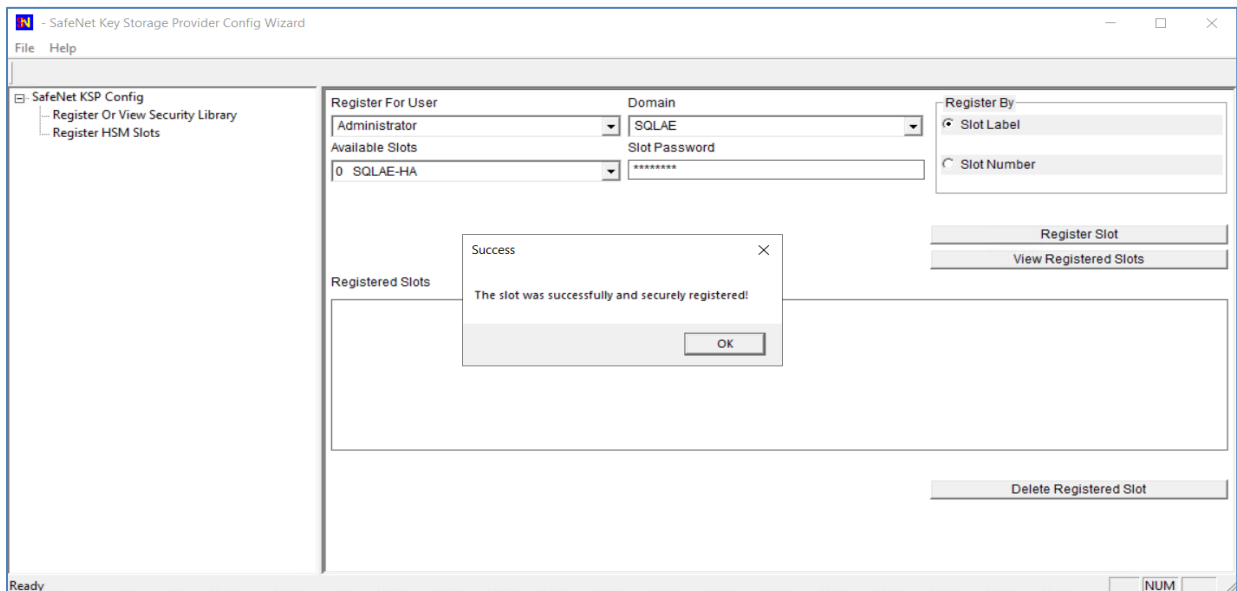
For Example: `C:\Program Files\SafeNet\LunaClient\KSP>KspConfig.exe`

3. Double-click **Register or View Security Library** on the left side of the pane.
4. Browse the library `<Luna Client installation Directory>\cryptoki.dll` library and click **Register**. On successful registration, you will see the following message: **"Success registering the security library"**.

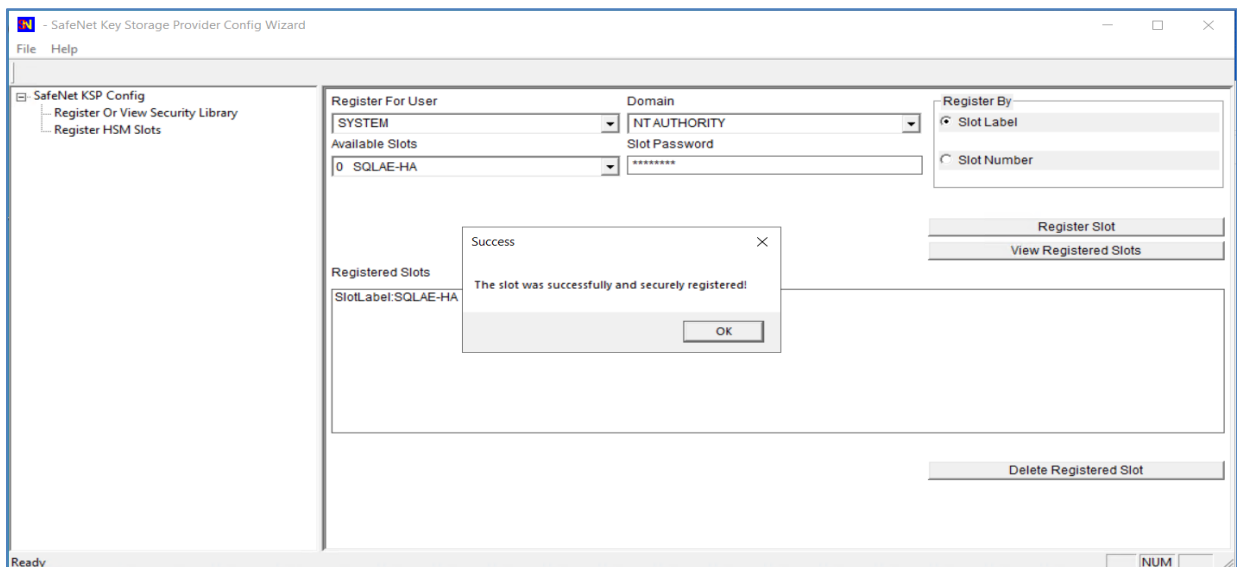


5. Double-click **Register HSM Slots** on the left side of the pane.
 - a. Click the **Register for User** drop-down menu and select the User.
 - b. Click the **Domain** drop-down and select your domain.
 - c. Click the **Available Slots** drop-down and select the partition.
 - d. Enter the partition password in **Slot Password** field.

- e. Click **Register Slot** to register the slot for Domain\User. On successful registration, a message "**The slot was successfully and securely registered**" will appear on screen.



6. Double-click **Register HSM Slots** on the left side of the pane.
 - a. Click the **Register for User** drop-down menu and select **SYSTEM**.
 - b. Click the **Domain** drop-down and select **NT AUTHORITY**.
 - c. Open the **Available Slots** drop-down and select the partition.
 - d. Enter the partition password in **Slot Password** field.
 - e. Click **Register Slot** to register the slot for NT_AUTHORITY\SYSTEM. On successful registration, a message "**The slot was successfully and securely registered**" will appear on screen.

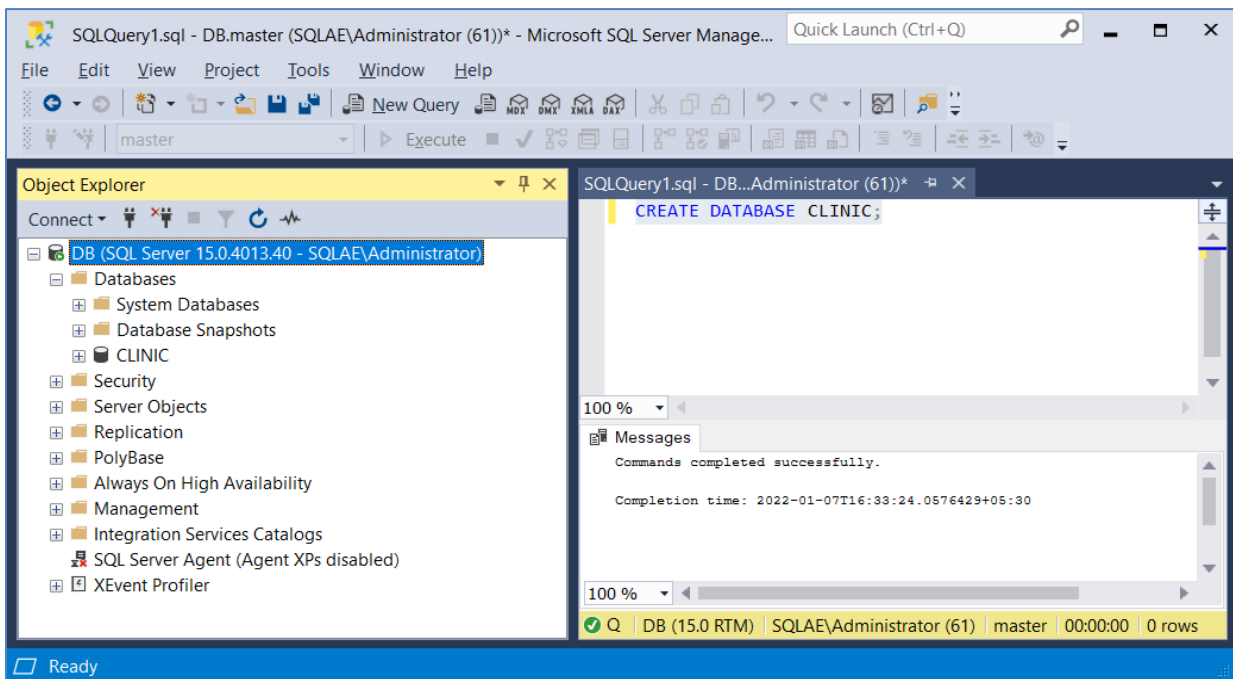


Create a Sample Database

For this demonstration, a sample database is need to be created on DB server. However, an existing database and table can also be used to encrypt required columns data. To create a sample database and a table for column encryption, log in to the SQL **DB** server and perform below steps:

1. Open the SQL Server Management Studio and connect to database server.
2. Create a sample database, such as **Clinic**.

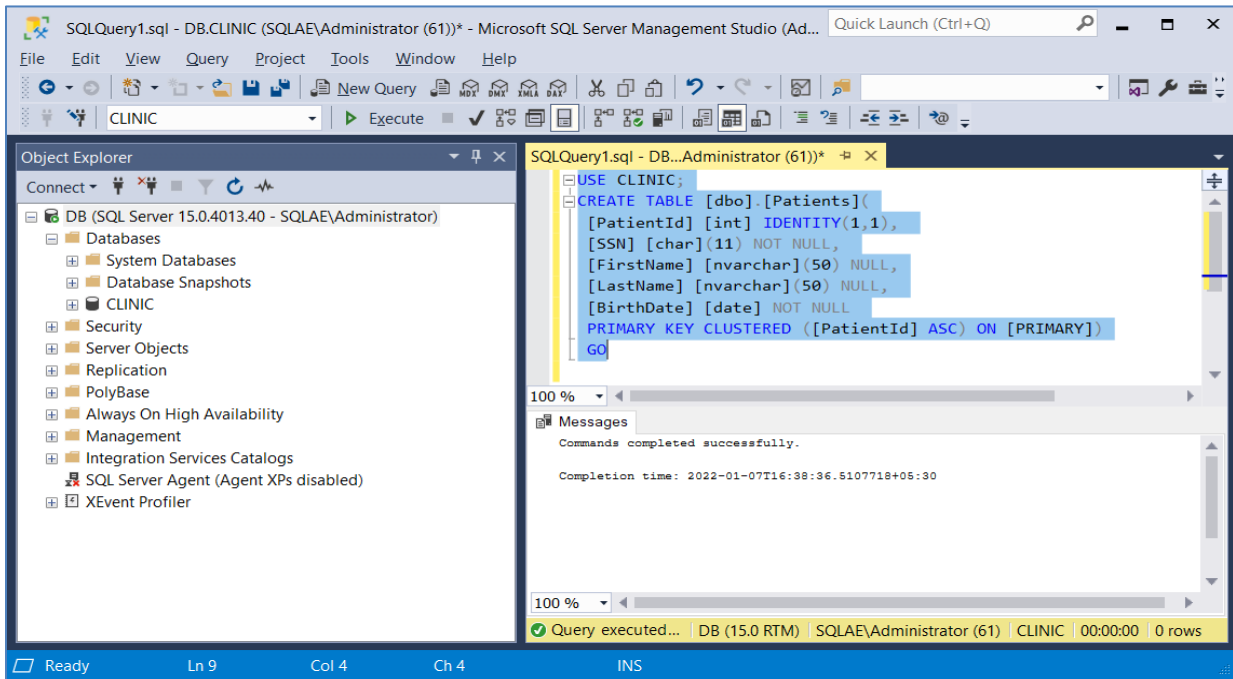
```
CREATE DATABASE CLINIC;
```



3. Create a sample table with few columns to perform column encryption.

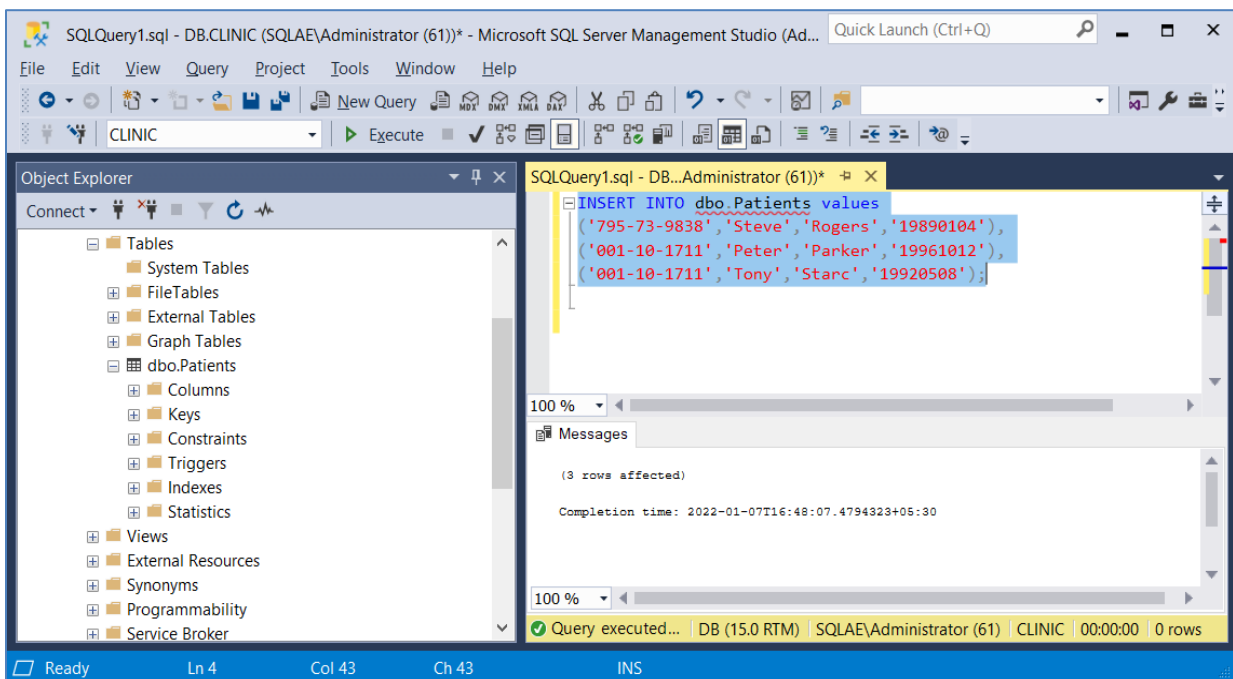
```
USE CLINIC;

CREATE TABLE [dbo].[Patients](
  [PatientId] [int] IDENTITY(1,1),
  [SSN] [char](11) NOT NULL,
  [FirstName] [nvarchar](50) NULL,
  [LastName] [nvarchar](50) NULL,
  [BirthDate] [date] NOT NULL
  PRIMARY KEY CLUSTERED ([PatientId] ASC) ON [PRIMARY])
GO
```



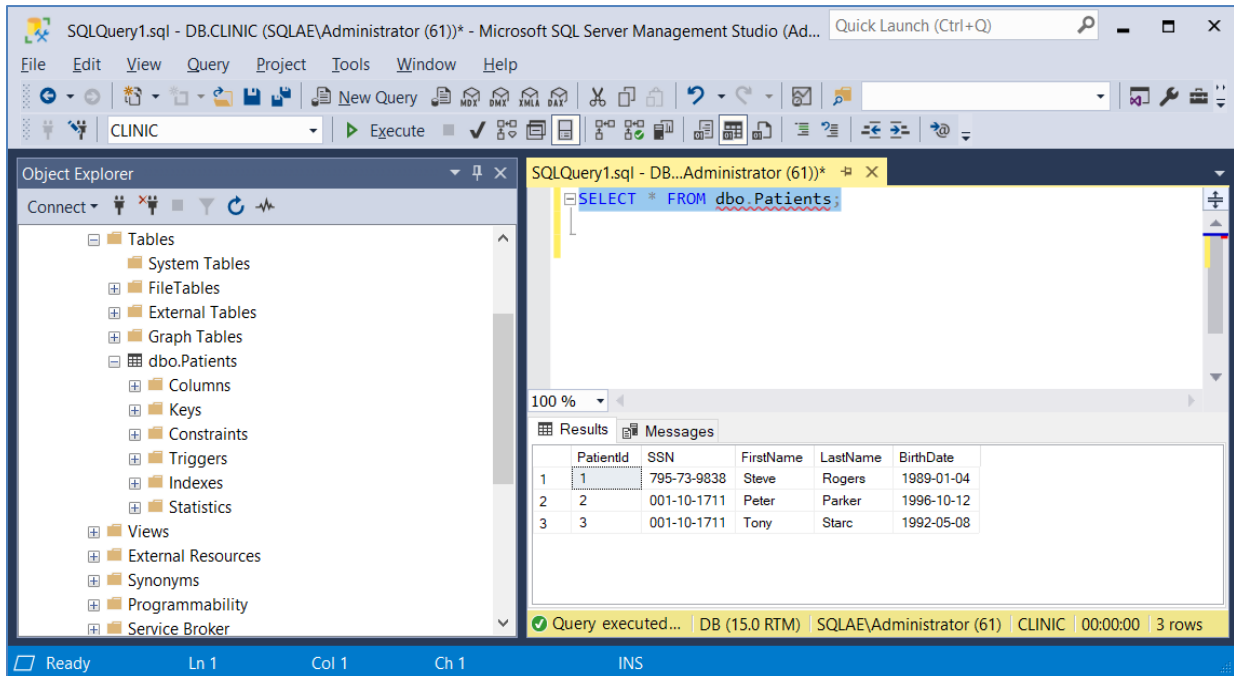
4. Insert some data in the table.

```
INSERT INTO dbo.Patients values
('795-73-9838','Steve','Rogers','19890104'),
('001-10-1711','Peter','Parker','19961012'),
('001-10-1711','Tony','Starc','19920508');
```



- View the data is saved and off course it is unencrypted at this stage.

```
SELECT * FROM dbo.Patients;
```



- Disconnect the database and log off from the **DB Server**.

Generate Column Master Key

Column master keys are key-protecting keys used to encrypt column encryption keys. Column master keys must be stored in a trusted key store, such as Windows Certificate Store, Azure Key Vault, or a hardware security module. The database only contains metadata about column master keys (the type of key store and location). To ensure Always Encrypted is effective, never generate column master keys or column encryption keys on a computer hosting your database. Instead, generate the keys on a separate computer, which is either dedicated for key management, or is a machine hosting applications that will need access to the keys anyway.

For this demonstration, a client machine will be used to generate the column master key and column encryption key. To generate the column master key, perform the following steps on a client machine separate from DB server:

- Log on to the **CL1** as an administrator where Luna Client is installed and SafeNet KSP is already registered with the HSM partition.
- Open the Window PowerShell app and click Run as administrator.

3. Before proceeding to generate the keys, install the **SqlServer** powershell module.

```
Install-Module -Name SqlServer -AllowClobber
```

```
PS C:\Users\Administrator> Install-Module -Name SqlServer -AllowClobber

NuGet provider is required to continue
PowerShellGet requires NuGet provider version '2.8.5.201' or newer to interact with NuGet-based repositories. The NuGet provider must be available in 'C:\Program
Files\PackageManagement\ProviderAssemblies' or 'C:\Users\Administrator\AppData\Local\PackageManagement\ProviderAssemblies'. You can also install the NuGet provider
by running 'Install-PackageProvider -Name NuGet -MinimumVersion 2.8.5.201 -Force'. Do you want PowerShellGet to install and import the NuGet provider now?
[Y] Yes [N] No [S] Suspend [?] Help (default is "Y"): Y

Untrusted repository
You are installing the modules from an untrusted repository. If you trust this repository, change its InstallationPolicy value by running the Set-PSRepository
cmdlet. Are you sure you want to install the modules from 'PSGallery'?
[Y] Yes [A] Yes to All [N] No [L] No to All [S] Suspend [?] Help (default is "N"): Y
PS C:\Users\Administrator>
```

4. Ensure that the SqlServer module is installed successfully. You will see output similar to the following sample:

```
Get-Module SqlServer -ListAvailable
```

```
PS C:\Users\Administrator> Get-Module SqlServer -ListAvailable

Directory: C:\Program Files\WindowsPowerShell\Modules

ModuleType Version Name ExportedCommands
-----
Script 21.1.18256 SqlServer {Add-RoleMember, Add-SqlAvailabilityDatabase, Add-SqlAvailabilityGroupListenerStaticIp, Add-SqlAzureAu...}

PS C:\Users\Administrator>
```

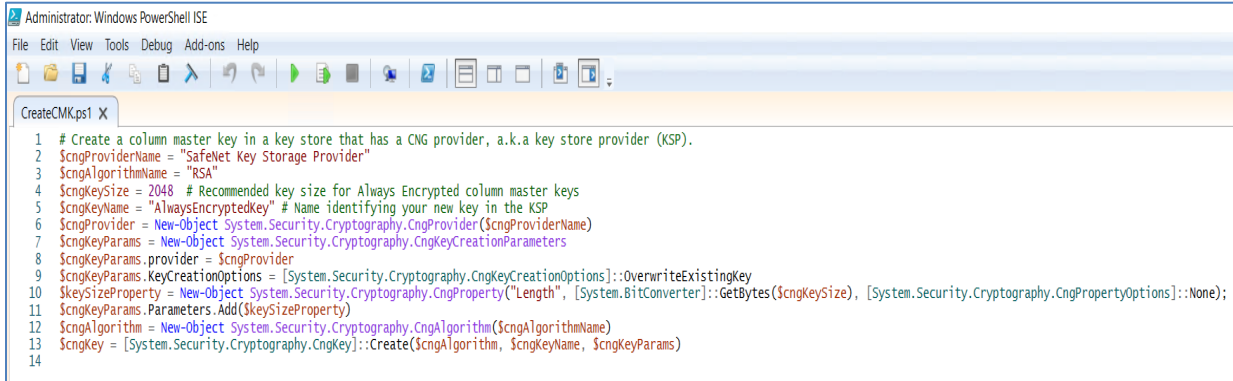
5. Create a column master key on HSM partition using the script provided below. Copy and paste the contents provided here in a file and save the file as .ps1 extension (for example, CreateCMK.ps1).

Create a column master key in a Hardware key store that has a CNG provider, a.k.a key store provider (KSP).

```
$cngProviderName = "SafeNet Key Storage Provider"
$cngAlgorithmName = "RSA"
$cngKeySize = 2048 # Recommended key size for Always Encrypted column master keys
$cngKeyName = "AlwaysEncryptedKey" # 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)
```



```
Administrator: Windows PowerShell ISE
File Edit View Tools Debug Add-ons Help
CreateCMK.ps1 X
1 # Create a column master key in a key store that has a CNG provider, a.k.a key store provider (KSP).
2 $cngProviderName = "SafeNet Key Storage Provider"
3 $cngAlgorithmName = "RSA"
4 $cngKeySize = 2048 # Recommended key size for Always Encrypted column master keys
5 $cngKeyName = "AlwaysEncryptedKey" # Name identifying your new key in the KSP
6 $cngProvider = New-Object System.Security.Cryptography.CngProvider($cngProviderName)
7 $cngKeyParams = New-Object System.Security.Cryptography.CngKeyCreationParameters
8 $cngKeyParams.provider = $cngProvider
9 $cngKeyParams.KeyCreationOptions = [System.Security.Cryptography.CngKeyCreationOptions]::OverwriteExistingKey
10 $keySizeProperty = New-Object System.Security.Cryptography.CngProperty("Length", [System.BitConverter]::GetBytes($cngKeySize), [System.Security.Cryptography.CngPropertyOptions]::None);
11 $cngKeyParams.Parameters.Add($keySizeProperty)
12 $cngAlgorithm = New-Object System.Security.Cryptography.CngAlgorithm($cngAlgorithmName)
13 $cngKey = [System.Security.Cryptography.CngKey]::Create($cngAlgorithm, $cngKeyName, $cngKeyParams)
14
```

6. Run the script to generate the CMK on HSM partition.

```
.\CreateCMK.ps1
```

```
PS C:\Users\Administrator> .\CreateCMK.ps1
PS C:\Users\Administrator>
```

7. Run the LunaCM utility provided with the Luna Client and check the CMK has been generated on HSM Partition.

```
& 'C:\Program Files\SafeNet\LunaClient\lunacm.exe'
role login -n co
par con
```

```
lunacm:> par con

The 'Crypto Officer' is currently logged in. Looking for objects
accessible to the 'Crypto Officer'.

Object list:

Label:          AlwaysEncryptedKey
Handle:         216
Object Type:    Private Key
Usage Limit:    none
Object UID:     5005000052000025b990800

Label:          AlwaysEncryptedKey
Handle:         215
Object Type:    Public Key
Usage Limit:    none
Object UID:     4f05000052000025b990800

Number of objects: 2

Command Result : No Error
```

8. Execute the following command to import the SqlServer module in PowerShell.

```
Import-Module "SqlServer"
```

```
PS C:\Users\Administrator> Import-Module "SqlServer"
PS C:\Users\Administrator> █
```

9. Now connect to the database and create column master key metadata in the database. Copy and paste the contents provided here in a file and save the with file .ps1 extension. For example: CMKMetadata.ps1.

NOTE: Replace the <server name> and <database name> with actual server name and database in your environment.

Connect to your database.

```
$serverName = "<server name>"
$databaseName = "<database name>"
```

Change the authentication method in the connection string, if needed.

```
$connStr = "Server = " + $serverName + "; Database = " + $databaseName +
"; Integrated Security = True"
$database = Get-SqlDatabase -ConnectionString $connStr
```

Create a SqlColumnMasterKeySettings object for your column master key.

```
$cngProviderName = "SafeNet Key Storage Provider"
$cngKeyName = "AlwaysEncryptedKey" # Your CMK key label on HSM Partition.
$cmkSettings = New-SqlCngColumnMasterKeySettings -CngProviderName
$cngProviderName -KeyName $cngKeyName
```

Create column master key metadata in the database.

```
$cmkName = "CMK1"

New-SqlColumnMasterKey -Name $cmkName -InputObject $database -
ColumnMasterKeySettings $cmkSettings
```

```
Administrator: Windows PowerShell ISE
File Edit View Tools Debug Add-ons Help
Untitled1.ps1 CMKMetadata.ps1 X
1 # Connect to your database.
2 $serverName = "DB"
3 $databaseName = "Clinic"
4 # Change the authentication method in the connection string, if needed.
5 $connStr = "Server = " + $serverName + "; Database = " + $databaseName + "; Integrated Security = True"
6 $database = Get-SqlDatabase -ConnectionString $connStr
7 # Create a SqlColumnMasterKeySettings object for your column master key.
8 $cngProviderName = "SafeNet Key Storage Provider"
9 $cngKeyName = "AlwaysEncryptedKey" # Your CMK key label on HSM Partition.
10 $cmkSettings = New-SqlCngColumnMasterKeySettings -CngProviderName $cngProviderName -KeyName $cngKeyName
11 # Create column master key metadata in the database.
12 $cmkName = "CMK1"
13 New-SqlColumnMasterKey -Name $cmkName -InputObject $database -ColumnMasterKeySettings $cmkSettings
14
15
```

10. Run the script to generate the CMK Metadata in sample database created.

```
.\CMKMetadata.ps1
```

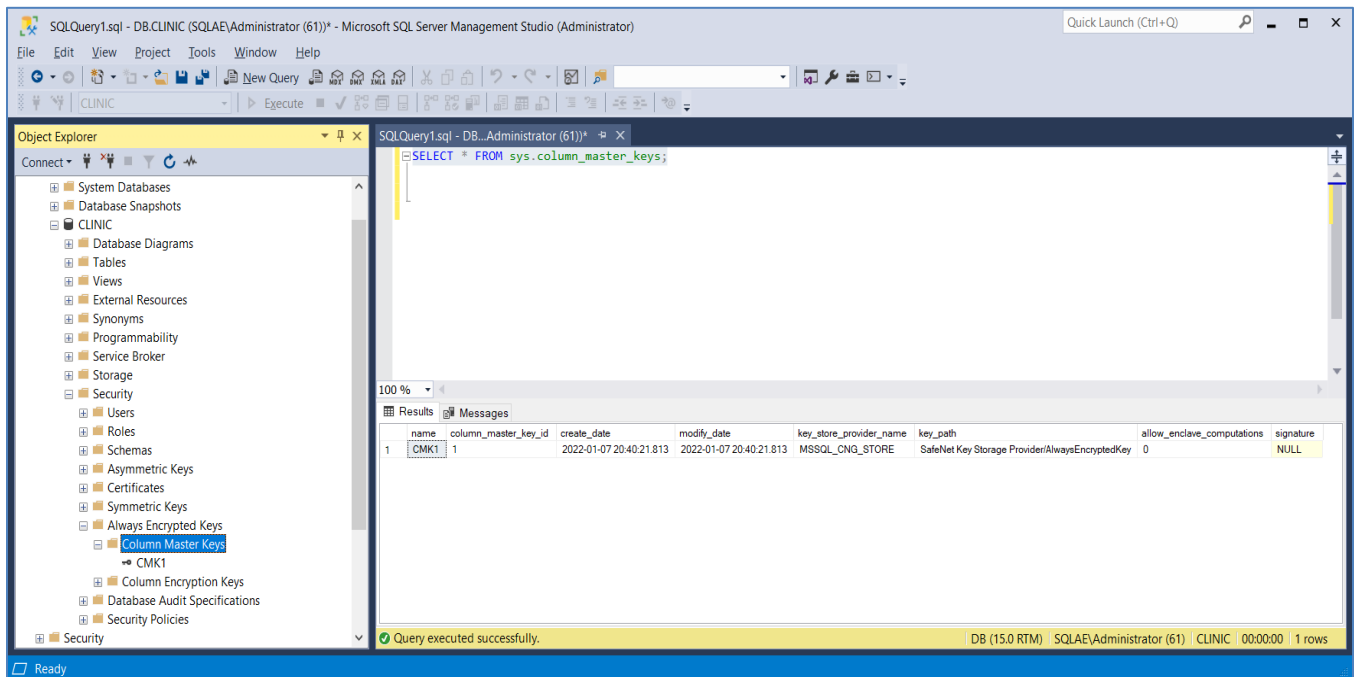
```
PS C:\Users\Administrator> .\CMKMetadata.ps1

Name
----
CMK1

PS C:\Users\Administrator> _
```

11. Connect to the **DB** server and run the following command to see the CMK metadata created in the database.

```
SELECT * FROM sys.column_master_keys;
```



Generate Column Encryption Key

Column encryption keys are content-encryption keys used to encrypt data. As the name implies, you use column encryption keys to encrypt data in database columns. You can encrypt 1 or more columns with the same column encryption key, or you can use multiple column encryption keys depending on your application requirements. The column encryption keys are themselves encrypted, and only the encrypted values of the column encryption keys are stored in the database (as part of the column encryption key metadata).

1. Log on to the **CL1** as an administrator where CMK is generated.
2. Open the Window PowerShell app and click Run as administrator.

- Now connect to the database and create column encryption key encrypted by column master key and store the encrypted CEK metadata in the database. Copy and paste the contents provided here in a file and save the file with .ps1 extension. For example: **CEKMetadata.ps1**

NOTE: Replace the <server name> and <database name> with actual server name and database in your environment.

Connect to your database.

```
$serverName = "<server name>"
$databaseName = "<database name>"
```

Change the authentication method in the connection string, if needed.

```
$connStr = "Server = " + $serverName + "; Database = " + $databaseName +
"; Integrated Security = True"
$database = Get-SqlDatabase -ConnectionString $connStr
```

Generate a column encryption key, encrypt it with the column master key and create column encryption key metadata in the database.

```
$cmkName = "CMK1"
$cekName = "CEK1"
New-SqlColumnEncryptionKey -Name $cekName -InputObject $database -
ColumnMasterKey $cmkName
```

The screenshot shows the Windows PowerShell ISE interface. The title bar reads "Administrator: Windows PowerShell ISE". The menu bar includes File, Edit, View, Tools, Debug, Add-ons, and Help. The toolbar contains various icons for file operations and execution. The script editor shows the following code:

```
1 # Connect to your database.
2 $serverName = "DB"
3 $databaseName = "Clinic"
4 # Change the authentication method in the connection string, if needed.
5 $connStr = "Server = " + $serverName + "; Database = " + $databaseName + "; Integrated Security = True"
6 $database = Get-SqlDatabase -ConnectionString $connStr
7 # Generate a column encryption key, encrypt it with the column master key and create column encryption key metadata in the database.
8 $cmkName = "CMK1"
9 $cekName = "CEK1"
10 New-SqlColumnEncryptionKey -Name $cekName -InputObject $database -ColumnMasterKey $cmkName
11
```

- Run the script to generate the CEK Metadata in sample database created.

```
.\CEKMetadata.ps1
```

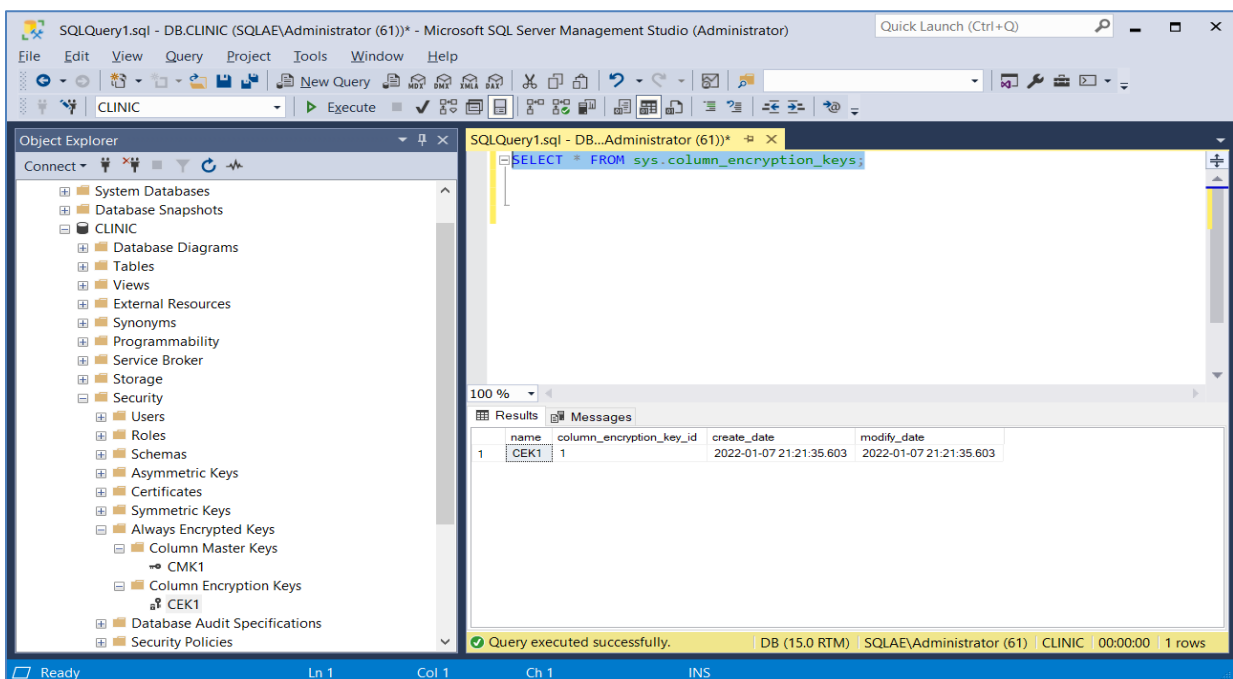
```
PS C:\Users\Administrator> .\CEKMetadata.ps1

Name
----
CEK1

PS C:\Users\Administrator>
```

5. Connect to the DB server and run the following command to see the CMK metadata created in the database.

```
SELECT * FROM sys.column_encryption_keys;
```



Encrypt Columns using Always Encrypted Keys

The following example demonstrates setting the target encryption configuration for a couple of columns using the offline approach. If either column isn't already encrypted, it will be encrypted. If any column is already encrypted using a different key and/or a different encryption type, it will be decrypted and then re-encrypted with the specified target key/type.

SQL Server supports Encryption Type values are one of the following:

- **Deterministic:** Same encrypted values for similar data.
- **Randomized:** Different encrypted value for same data.
- **Plaintext** (only available to revert encrypted columns to an unencrypted state).

1. Log on to the **CL1** as an administrator.
2. Copy and paste the following contents in the file and save the file with **.ps1** extension (for example, **encrypt.ps1**).

NOTE: Replace the <server name> and <database name> with actual server name and database in your environment.

Import the SqlServer module.

```
Import-Module "SqlServer"
```

Connect to your database.

```
$serverName = "<server name>"
```

```
$databaseName = "<database name>"
```

Change the authentication method in the connection string, if needed.

```
$connStr = "Server = " + $serverName + "; Database = " + $databaseName +
"; Integrated Security = True"
```

```
$database = Get-SqlDatabase -ConnectionString $connStr
```

Encrypt the selected columns (or re-encrypt, if they are already encrypted using keys/encrypt types, different than the specified keys/types.

```
$ces = @()
```

```
$ces += New-SqlColumnEncryptionSettings -ColumnName "dbo.Patients.SSN" -
EncryptionType "Deterministic" -EncryptionKey "CEK1"
```

```
$ces += New-SqlColumnEncryptionSettings -ColumnName
"dbo.Patients.BirthDate" -EncryptionType "Randomized" -EncryptionKey
"CEK1"
```

```
Set-SqlColumnEncryption -InputObject $database -ColumnEncryptionSettings
$ces -LogFileDirectory .
```

The screenshot shows the Windows PowerShell ISE interface. The title bar reads "Administrator: Windows PowerShell ISE". The menu bar includes "File", "Edit", "View", "Tools", "Debug", "Add-ons", and "Help". The toolbar contains various icons for file operations and execution. Two tabs are open: "Untitled1.ps1" and "encrypt.ps1". The "encrypt.ps1" tab is active, displaying the following PowerShell script:

```

1 # Import the SqlServer module.
2 Import-Module "SqlServer"
3
4 # Connect to your database.
5 $serverName = "DB"
6 $databaseName = "Clinic"
7 # Change the authentication method in the connection string, if needed.
8 $connStr = "Server = " + $serverName + "; Database = " + $databaseName + "; Integrated Security = True"
9 $database = Get-SqlDatabase -ConnectionString $connStr
10
11 # Encrypt the selected columns (or re-encrypt, if they are already encrypted using keys/encrypt types, different than the specified keys/types.
12 $ces = @()
13 $ces += New-SqlColumnEncryptionSettings -ColumnName "dbo.Patients.SSN" -EncryptionType "Deterministic" -EncryptionKey "CEK1"
14 $ces += New-SqlColumnEncryptionSettings -ColumnName "dbo.Patients.BirthDate" -EncryptionType "Randomized" -EncryptionKey "CEK1"
15 Set-SqlColumnEncryption -InputObject $database -ColumnEncryptionSettings $ces -LogFileDirectory .

```

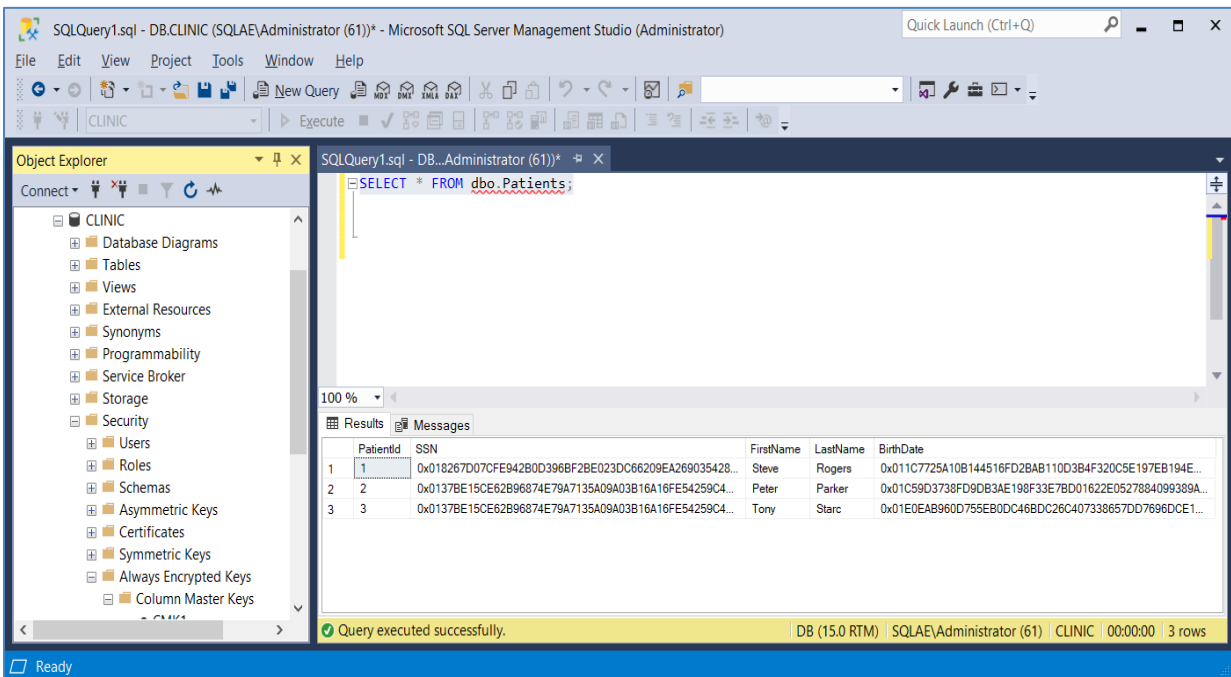
- Run the script in PowerShell to encrypt the column in the sample table.

```
.\encrypt.ps1
```

```
PS C:\Users\Administrator> .\encrypt.ps1
PS C:\Users\Administrator>
```

- Connect to the DB server and run the following command to verify the columns which must be encrypted now.

```
SELECT * FROM dbo.Patients;
```



View Always Encrypted Data

Data encrypted using Always Encrypted keys can be decrypted only at the client site by the client, who has access to the Column Master Key created and stored on HSM partition. Database Administrator and other users cannot see the encrypted data if they don't have access to CMK created on Luna HSM.

To view the Always Encrypted data, perform the below steps at the Client server who have access to the CMK generated on HSM partition.

- Log on to the **CL1** as an administrator.
- Copy and paste the following contents in the file and save the file with .ps1 extension (for example, viewdata.ps1):

NOTE: Replace the <server name> and <database name> with actual server name and database in your environment.

Import the SqlServer module.

```
Import-Module "SqlServer"
```

Connect to your database.

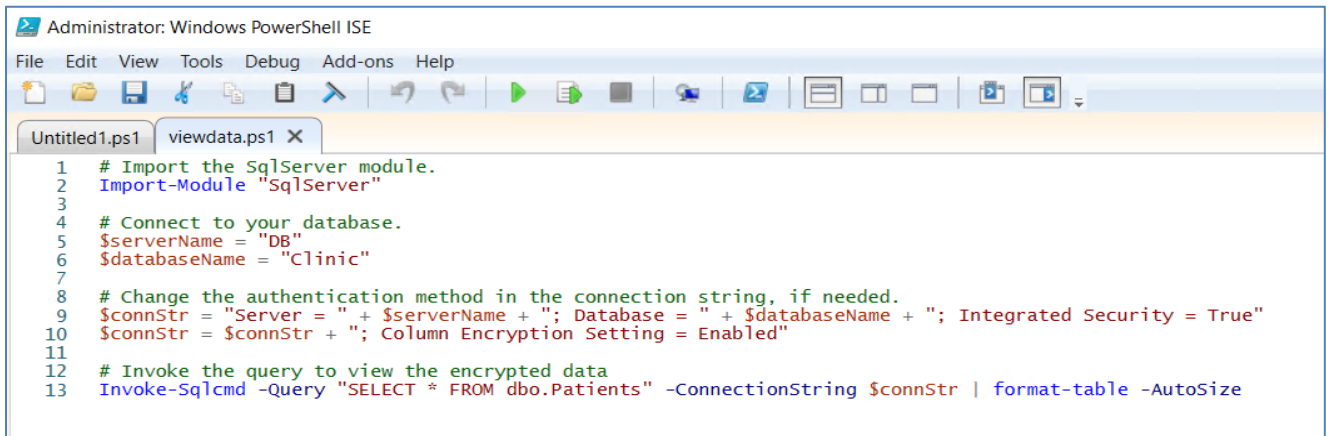
```
$serverName = "<server name>"
$databaseName = "<database name>"
```

Change the authentication method in the connection string, if needed.

```
$connStr = "Server = " + $serverName + "; Database = " + $databaseName +
"; Integrated Security = True"
$connStr = $connStr + "; Column Encryption Setting = Enabled"
```

Invoke the query to view the encrypted data

```
Invoke-Sqlcmd -Query "SELECT * FROM dbo.Patients" -ConnectionString
$connStr | format-table -AutoSize
```

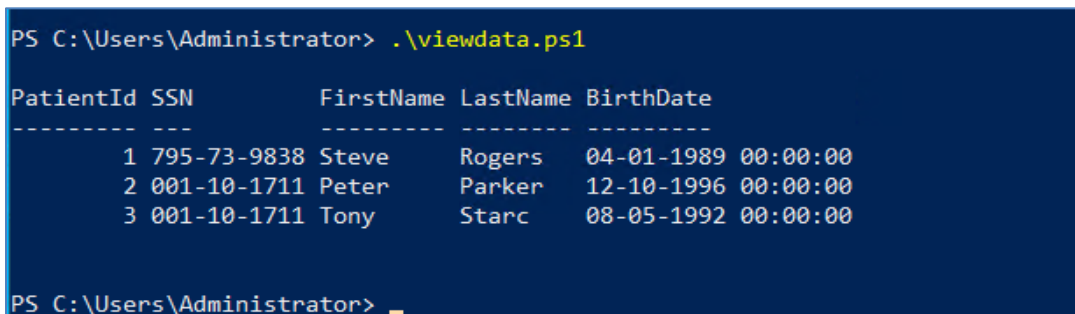


The screenshot shows the Windows PowerShell ISE interface. The title bar reads "Administrator: Windows PowerShell ISE". The menu bar includes File, Edit, View, Tools, Debug, Add-ons, and Help. The toolbar contains various icons for file operations and execution. The script editor shows the following code:

```
1 # Import the SqlServer module.
2 Import-Module "SqlServer"
3
4 # Connect to your database.
5 $serverName = "DB"
6 $databaseName = "Clinic"
7
8 # Change the authentication method in the connection string, if needed.
9 $connStr = "Server = " + $serverName + "; Database = " + $databaseName + "; Integrated Security = True"
10 $connStr = $connStr + "; Column Encryption Setting = Enabled"
11
12 # Invoke the query to view the encrypted data
13 Invoke-Sqlcmd -Query "SELECT * FROM dbo.Patients" -ConnectionString $connStr | format-table -AutoSize
```

- Run the script in PowerShell to encrypt the column in the sample table.

```
.\viewdata.ps1
```



The screenshot shows a PowerShell terminal window with the following output:

```
PS C:\Users\Administrator> .\viewdata.ps1

PatientId SSN          FirstName LastName BirthDate
-----
1 795-73-9838 Steve    Rogers   04-01-1989 00:00:00
2 001-10-1711 Peter    Parker  12-10-1996 00:00:00
3 001-10-1711 Tony     Starc   08-05-1992 00:00:00

PS C:\Users\Administrator>
```

Viewing Always Encrypted Data on Multiple Clients

To view always encrypted data on multiple clients, each client must have access to HSM partition where CMK is being generated and SafeNet KSP must be registered on the client, if not, follow the instructions mentioned in [Configure SafeNet KSP](#).

1. Log on to the CL2 as an administrator where Luna Client is installed and SafeNet KSP is registered with the HSM partition.
2. Open the Command Prompt and run the ksputil to see if the CMK, created on HSM partition by another client system (CL1), is accessible.

```
C:\Program Files\SafeNet\LunaClient\KSP>ksputil.exe listKeys /s 0 /user
```

Where /s <num> specifies the Slot ID of partition and /user specifies current logged in user. When the command get executed, you will be prompted to provide a Crypto Officer Password.

```
Administrator: C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19043.1415]
(c) Microsoft Corporation. All rights reserved.

C:\Program Files\SafeNet\LunaClient\KSP>ksputil.exe listKeys /s 0 /user
ksputil.exe (64-bit) v10.4.0-417. Copyright (c) 2021 SafeNet. All rights reserved.

This Servers Host Name is: CL2.SQLAE.com and the logged on user is: Administrator@SQLAE
Enter challenge for slot '0' :*****
UserKey:           AlwaysEncryptedKey           Handle: 2000001
UserKey:           AlwaysEncryptedKey           Handle: 2000002
```

If the CMK generated from another client is accessible, Always Encrypted data can be decrypted at this client.

3. Open the Windows PowerShell app and click Run as administrator.
4. Before requesting the encrypted data from database server, install the SqlServer powershell module.

```
Install-Module -Name SqlServer -AllowClobber
```

```
PS C:\Users\Administrator> Install-Module -Name SqlServer -AllowClobber
NuGet provider is required to continue
PowerShellGet requires NuGet provider version '2.8.5.201' or newer to interact with NuGet-based repositories. The NuGet provider must be available in 'C:\Program Files\PackageManagement\ProviderAssemblies' or 'C:\Users\Administrator\AppData\Local\PackageManagement\ProviderAssemblies'. You can also install the NuGet provider by running 'Install-PackageProvider -Name NuGet -MinimumVersion 2.8.5.201 -Force'. Do you want PowerShellGet to install and import the NuGet provider now?
[Y] Yes [N] No [S] Suspend [?] Help (default is "Y"): Y

Untrusted repository
You are installing the modules from an untrusted repository. If you trust this repository, change its InstallationPolicy value by running the Set-PSRepository cmdlet. Are you sure you want to install the modules from 'PSGallery'?
[Y] Yes [A] Yes to All [N] No [L] No to All [S] Suspend [?] Help (default is "N"): Y
PS C:\Users\Administrator>
```

5. Ensure that the SqlServer module is installed successfully. You will see output similar to the example below.

```
Get-Module SqlServer -ListAvailable
```

```
PS C:\Users\Administrator> Get-Module SqlServer -ListAvailable
Directory: C:\Program Files\WindowsPowerShell\Modules

ModuleType Version Name ExportedCommands
-----
Script 21.1.18256 SqlServer {Add-RoleMember, Add-SqlAvailabilityDatabase, Add-SqlAvailabilityGroupListenerStaticIp, Add-SqlAzureAu...}
```

6. Copy and paste the following contents in the file and save the file with .ps1 extension. For Example: **viewdata.ps1**

NOTE: Replace the <server name> and <database name> with actual server name and database in your environment.

Import the SqlServer module.

```
Import-Module "SqlServer"
```

Connect to your database.

```
$serverName = "<server name>"
```

```
$databaseName = "<database name>"
```

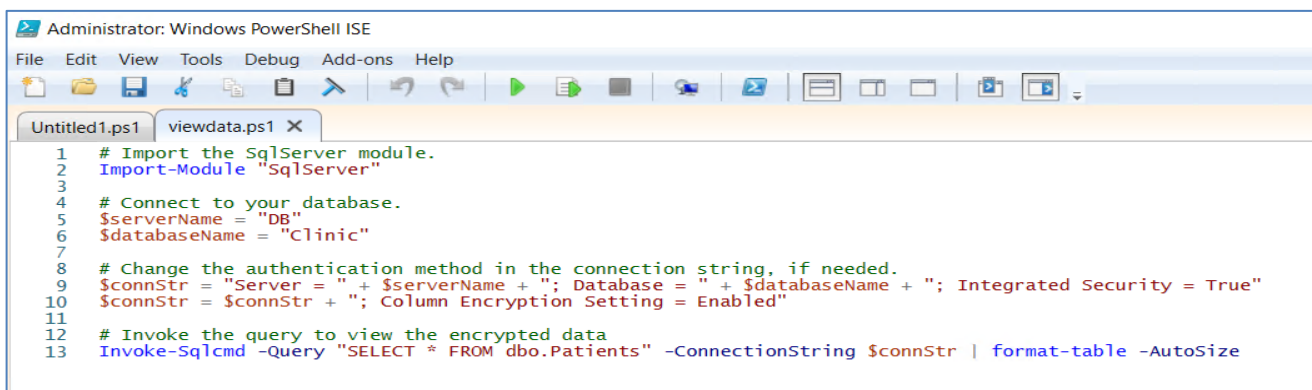
Change the authentication method in the connection string, if needed.

```
$connStr = "Server = " + $serverName + "; Database = " + $databaseName +  
"; Integrated Security = True"
```

```
$connStr = $connStr + "; Column Encryption Setting = Enabled"
```

Invoke the query to view the encrypted data.

```
Invoke-Sqlcmd -Query "SELECT * FROM dbo.Patients" -ConnectionString  
$connStr | format-table -AutoSize
```



The screenshot shows the Windows PowerShell ISE interface. The title bar reads "Administrator: Windows PowerShell ISE". The menu bar includes "File", "Edit", "View", "Tools", "Debug", "Add-ons", and "Help". The toolbar contains various icons for file operations and execution. Two tabs are open: "Untitled1.ps1" and "viewdata.ps1". The "viewdata.ps1" tab is active, displaying the following PowerShell script:

```
1 # Import the SqlServer module.  
2 Import-Module "SqlServer"  
3  
4 # Connect to your database.  
5 $serverName = "DB"  
6 $databaseName = "Clinic"  
7  
8 # Change the authentication method in the connection string, if needed.  
9 $connStr = "Server = " + $serverName + "; Database = " + $databaseName + "; Integrated Security = True"  
10 $connStr = $connStr + "; Column Encryption Setting = Enabled"  
11  
12 # Invoke the query to view the encrypted data  
13 Invoke-Sqlcmd -Query "SELECT * FROM dbo.Patients" -ConnectionString $connStr | format-table -AutoSize
```

7. Run the script in PowerShell to encrypt the column in the sample table.

```
.\viewdata.ps1
```

```
PS C:\Users\Administrator> .\viewdata.ps1

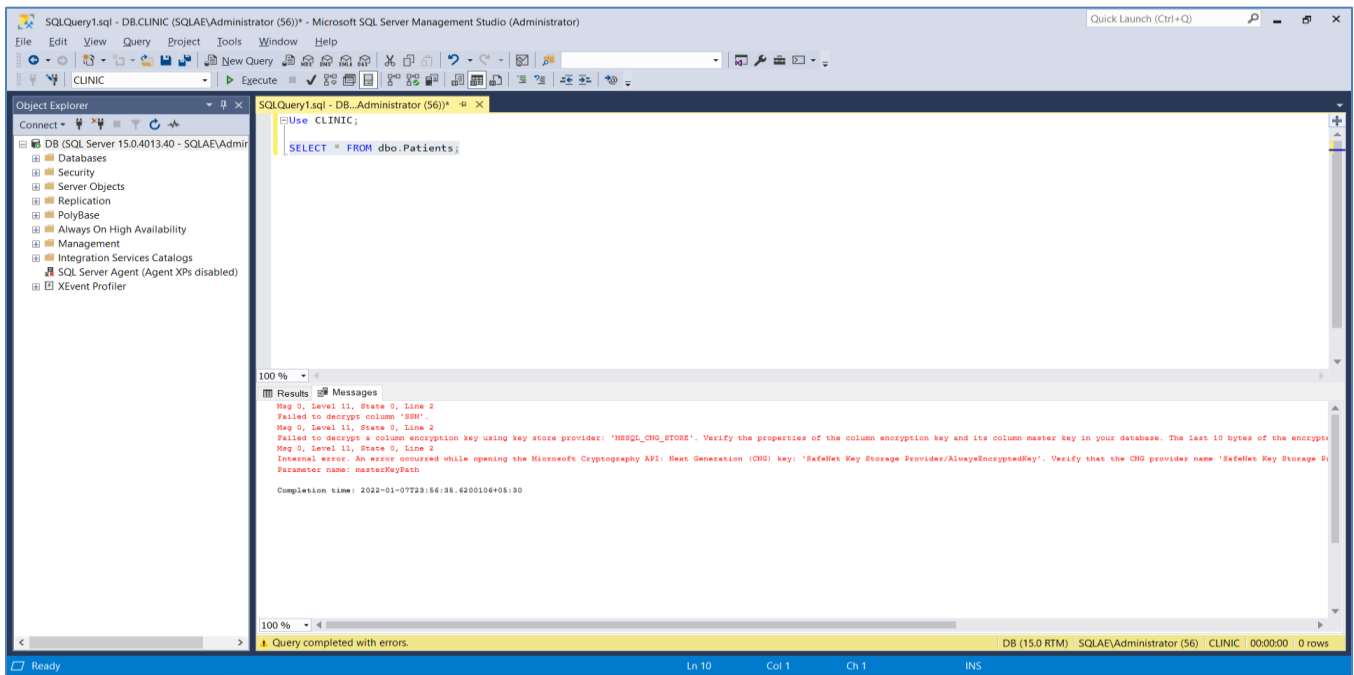
PatientId SSN          FirstName LastName BirthDate
-----
1 795-73-9838 Steve   Rogers   04-01-1989 00:00:00
2 001-10-1711 Peter   Parker   12-10-1996 00:00:00
3 001-10-1711 Tony    Starc    08-05-1992 00:00:00

PS C:\Users\Administrator>
```

The encrypted data is only accessible by the clients having access to CMK. CMK is accessible only if:

- Client is registered with the same Luna Partition.
- SafeNet KSP is registered using the same user on all clients.

The encrypted data can be viewed only by clients who have access to the Luna HSM partition and have registered SafeNet KSP with the users having access to the CMK. Any other users, including DBA, cannot view the encrypted data without access to the CMK even after using parametrized query and Column Encryption Setting=Enabled.



The SQL Server Always Encrypted integration using Luna HSM is completed and Column Master Key, used to encrypt Column Encryption Key, is securely stored on Luna HSM. In the next section, we will see how to rotate the always encrypted keys and decrypt the encrypted data to plaintext when column encryption is not required.

Rotate Always Encrypted Key

Rotating Always Encrypted Keys is the process of replacing an existing key with a new one. The key needs to be rotated if it has been compromised, or in order to comply with your organization's policies or compliance regulations that mandate cryptographic keys must be rotated on a regular basis.

Always Encrypted uses two types of keys, so there are two high-level key rotation workflows; rotating column master keys, and rotating column encryption keys.

Column encryption key rotation - involves decrypting data that is encrypted with the current key, and re-encrypting the data using the new column encryption key. Rotating a column encryption key requires access to both the keys and the database.

Column master key rotation - involves decrypting column encryption keys that are protected with the current column master key, re-encrypting them using the new column master key, and updating the metadata for both types of keys.

Rotating a Column Master Key (CMK)

The following script demonstrates replacing an existing column master key (CMK1) created initially using [Generate Column Master Key](#) with a new column master key (CMK2). To rotate the Column Master Key:

1. Log on to any Client (**CL1** or **CL2**) as a server administrator, copy and paste the following contents in a file, and then save the file with .ps1 extension (for example, rotateCMK.ps1).

NOTE: Replace the <server name> and <database name> with actual server name and database in your environment. In addition, provide cngKeyName as per your choice to recognize the new key in the KSP. The key will be generated on HSM partition with same key label specified here.

NOTE: It is recommended that you should not permanently delete the old column master key after the key rotation. Instead, the old column master key must be kept in the Luna HSM partition or any secured location such as Luna Backup HSM. If you restore your database from a backup file to a point in time before the new column master key was configured, you must need the old column master key to access the data restored.

Create a new column master key in a Hardware key store that has a CNG provider, a.k.a key store provider (KSP).

```
$cngProviderName = "SafeNet Key Storage Provider"
$cngAlgorithmName = "RSA"
$cngKeySize = 2048
$cngKeyName = "AlwaysEncryptedKey10012022" # 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)

# Import the SqlServer module.

Import-Module "SqlServer"

# Connect to your database.

$serverName = "<server name>"
$databaseName = "<database name>"

# Change the authentication method in the connection string, if needed.

$connStr = "Server = " + $serverName + "; Database = " + $databaseName +
"; Integrated Security = True"

$database = Get-SqlDatabase -ConnectionString $connStr

# Create a SqlColumnMasterKeySettings object for your new column master key.

$newCmkSettings = New-SqlCngColumnMasterKeySettings -CngProviderName
$cngProviderName -KeyName $cngKeyName

# Create a new column master key metadata in the database.

$newCmkName = "CMK2"

New-SqlColumnMasterKey -Name $newCmkName -InputObject $database -
ColumnMasterKeySettings $newCmkSettings

# Initiate the rotation from the current column master key to the new column master key.

$oldCmkName = "CMK1"

Invoke-SqlColumnMasterKeyRotation -SourceColumnMasterKeyName $oldCmkName -
TargetColumnMasterKeyName $newCmkName -InputObject $database

# Complete the rotation of the old column master key.

Complete-SqlColumnMasterKeyRotation -SourceColumnMasterKeyName $oldCmkName
-InputObject $database

# Remove the old column master key metadata.

Remove-SqlColumnMasterKey -Name $oldCmkName -InputObject $database

```

```

1 # Create a new column master key in a Hardware key store that has a CNG provider, a.k.a key store provider (KSP).
2 $cngProviderName = "SafeNet Key Storage Provider"
3 $cngAlgorithmName = "RSA"
4 $cngKeySize = 2048
5 $cngKeyName = "AlwaysEncryptedKey10012022" # Name identifying your new key in the KSP.
6 $cngProvider = New-Object System.Security.Cryptography.CngProvider($cngProviderName)
7 $cngKeyParams = New-Object System.Security.Cryptography.CngKeyCreationParameters
8 $cngKeyParams.provider = $cngProvider
9 $cngKeyParams.KeyCreationOptions = [System.Security.Cryptography.CngKeyCreationOptions]::OverwriteExistingKey
10 $keySizeProperty = New-Object System.Security.Cryptography.CngProperty("Length", [System.BitConverter]::GetBytes($cngKeySize), [System.Security.Cryptography.CngPropertyOptions]::)
11 $cngKeyParams.Parameters.Add($keySizeProperty)
12 $cngAlgorithm = New-Object System.Security.Cryptography.CngAlgorithm($cngAlgorithmName)
13 $cngKey = [System.Security.Cryptography.CngKey]::Create($cngAlgorithm, $cngKeyName, $cngKeyParams)
14
15 # Import the SqlServer module.
16 Import-Module "SqlServer"
17
18 # Connect to your database.
19 $serverName = "DB"
20 $dbName = "Clinic"
21 # Change the authentication method in the connection string, if needed.
22 $connStr = "Server = " + $serverName + "; Database = " + $dbName + "; Integrated Security = True"
23 $database = Get-SqlDatabase -ConnectionString $connStr
24
25 # Create a SqlColumnMasterKeySettings object for your new column master key.
26 $newCmkSettings = New-SqlCngColumnMasterKeySettings -CngProviderName $cngProviderName -KeyName $cngKeyName
27
28 # Create a new column master key metadata in the database.
29 $newCmkName = "CMK2"
30 New-SqlColumnMasterKey -Name $newCmkName -InputObject $database -ColumnMasterKeySettings $newCmkSettings
31
32 # Initiate the rotation from the current column master key to the new column master key.
33 $oldCmkName = "CMK1"
34 Invoke-SqlColumnMasterKeyRotation -SourceColumnMasterKeyName $oldCmkName -TargetColumnMasterKeyName $newCmkName -InputObject $database
35
36 # Complete the rotation of the old column master key.
37 Complete-SqlColumnMasterKeyRotation -SourceColumnMasterKeyName $oldCmkName -InputObject $database
38
39 # Remove the old column master key metadata.
40 Remove-SqlColumnMasterKey -Name $oldCmkName -InputObject $database
41

```

- Run the script in PowerShell to rotate the Column Master Key.

```
.\rotateCMK.ps1
```

```
PS C:\Users\Administrator> .\rotateCMK.ps1
```

```
Name
----
CMK2
```

```
PS C:\Users\Administrator> █
```

3. Confirm the newly generated column master key on HSM partition using LunaCM.

```
& 'C:\Program Files\SafeNet\LunaClient\lunacm.exe'  
role login -n co  
par con
```

```
lunacm:> par con
```

```
The 'Crypto Officer' is currently logged in. Looking for objects  
accessible to the 'Crypto Officer'.
```

```
Object list:
```

```
Label:      AlwaysEncryptedKey10012022  
Handle:     327  
Object Type: Private Key  
Usage Limit: none  
Object UID: ed060000520000025b990800
```

```
Label:      AlwaysEncryptedKey10012022  
Handle:     212  
Object Type: Public Key  
Usage Limit: none  
Object UID: ec060000520000025b990800
```

```
Label:      AlwaysEncryptedKey  
Handle:     216  
Object Type: Private Key  
Usage Limit: none  
Object UID: 50050000520000025b990800
```

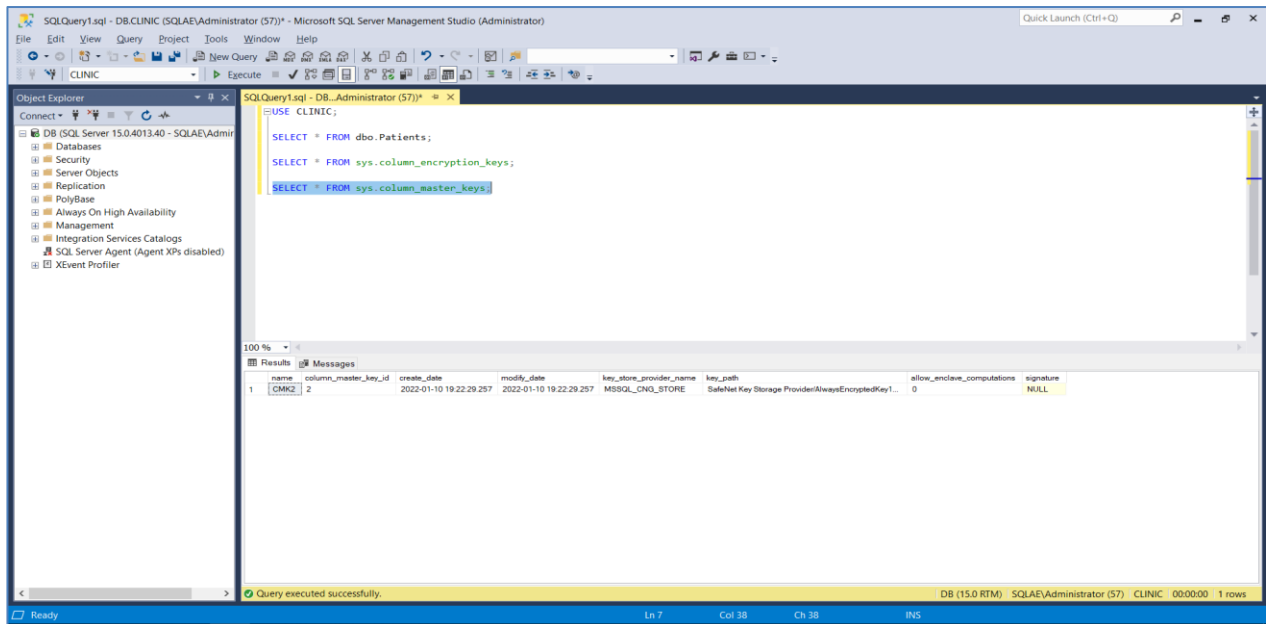
```
Label:      AlwaysEncryptedKey  
Handle:     215  
Object Type: Public Key  
Usage Limit: none  
Object UID: 4f050000520000025b990800
```

```
Number of objects: 4
```

```
Command Result : No Error
```

4. Connect to the **DB** server and confirm the new CMK metadata created in the database.

```
SELECT * FROM sys.column_master_keys;
```



The rotation of column master key is completed successfully and you can view the encrypted columns with new column master key using [View Always Encrypted Data](#).

Rotating a Column Encryption Key (CEK)

Rotating a column encryption key can take a long time if there are large tables containing columns encrypted with the key being rotated. Therefore, the organization needs to plan a column encryption key rotation carefully because during the rotation applications can't write to the impacted tables.

The below script demonstrates rotating a column encryption key. The script assumes, the target database contains some columns encrypted with a column encryption key, named CEK1 (to be rotated which was initially created in using [Generate Column Encryption Key](#)), which is protected using a column master key, named CMK2 (rotated CMK which in now used to encrypt the CEK1 after [Rotating a Column Master Key](#)). To rotate the Column Encryption Key:

1. Log on to any Client (**CL1** or **CL2**) as a server administrator, copy and paste the following contents in a file, and save the file with .ps1 extension (for example, **rotateCEK.ps1**).

NOTE: Replace the <server name> and <database name> with actual server name and database in your environment.

Import the SqlServer module.

```
Import-Module "SqlServer"
```

Connect to your database.

```
$serverName = "<server name>"
```

```

$databaseName = "<database name>"

# Change the authentication method in the connection string, if needed.

$connStr = "Server = " + $serverName + "; Database = " + $databaseName +
"; Integrated Security = True"

$database = Get-SqlDatabase -ConnectionString $connStr

# Generate a new column encryption key, encrypt it with the column master key and create column
encryption key metadata in the database.

$cmkName = "CMK2"
$newCekName = "CEK2"

New-SqlColumnEncryptionKey -Name $newCekName -InputObject $database -
ColumnMasterKey $cmkName

# Find all columns encrypted with the old column encryption key, and create a
SqlColumnEncryptionSetting object for each column.

$ces = @()
$oldCekName = "CEK1"
$tables = $database.Tables
for($i=0; $i -lt $tables.Count; $i++){
    $columns = $tables[$i].Columns
    for($j=0; $j -lt $columns.Count; $j++) {
        if($columns[$j].isEncrypted -and
$columns[$j].ColumnEncryptionKeyName -eq $oldCekName) {
            $threeColPartName = $tables[$i].Schema + "." +
$tables[$i].Name + "." + $columns[$j].Name

            $ces += New-SqlColumnEncryptionSettings -ColumnName
$tables[$i].Name + "." + $columns[$j].Name -EncryptionType
$columns[$j].EncryptionType -
EncryptionKey $newCekName
        }
    }
}

# Re-encrypt all columns, currently encrypted with the old column encryption key, using the new
column encryption key.

Set-SqlColumnEncryption -ColumnEncryptionSettings $ces -InputObject
$database -UseOnlineApproach -MaxDowntimeInSeconds 120 -LogFileDirectory
.

# Remove the old column encryption key metadata.

Remove-SqlColumnEncryptionKey -Name $oldCekName -InputObject $database

```

```

1 # Import the SqlServer module.
2 Import-Module "SqlServer"
3
4 # Connect to your database.
5 $serverName = "DB"
6 $databaseName = "Clinic"
7 # Change the authentication method in the connection string, if needed.
8 $connStr = "Server = " + $serverName + "; Database = " + $databaseName + "; Integrated Security = True"
9 $database = Get-SqlDatabase -ConnectionString $connStr
10
11 # Generate a new column encryption key, encrypt it with the column master key and create column encryption key metadata in the database.
12 $cmkName = "CMK2"
13 $newCekName = "CEK2"
14 New-SqlColumnEncryptionKey -Name $newCekName -InputObject $database -ColumnMasterKey $cmkName
15
16 # Find all columns encrypted with the old column encryption key, and create a SqlColumnEncryptionSetting object for each column.
17 $ces = @()
18 $oldCekName = "CEK1"
19 $tables = $database.Tables
20 For($i=0; $i -lt $tables.Count; $i++){
21     $columns = $tables[$i].Columns
22     for($j=0; $j -lt $columns.Count; $j++) {
23         if($columns[$j].isEncrypted -and $columns[$j].ColumnEncryptionKeyName -eq $oldCekName) {
24             $threeColPartName = $tables[$i].Schema + "." + $tables[$i].Name + "." + $columns[$j].Name
25             $ces += New-SqlColumnEncryptionSettings -ColumnName $threeColPartName -EncryptionType $columns[$j].EncryptionType -EncryptionKey $newCekName
26         }
27     }
28 }
29
30 # Re-encrypt all columns, currently encrypted with the old column encryption key, using the new column encryption key.
31 Set-SqlColumnEncryption -ColumnEncryptionSettings $ces -InputObject $database -UseOnlineApproach -MaxDowntimeInSeconds 120 -LogFileDirectory .
32
33 # Remove the old column encryption key metadata.
34 Remove-SqlColumnEncryptionKey -Name $oldCekName -InputObject $database
35

```

2. Run the script in PowerShell to rotate the Column Encryption Key.

```

.\rotateCEK.ps1

```

```

Migrating data
  Encrypting your data
  [
  ]
  Acquiring database model and preparing data migration.

```

```

PS C:\Users\Administrator> .\rotateCEK.ps1

Name
----
CEK2

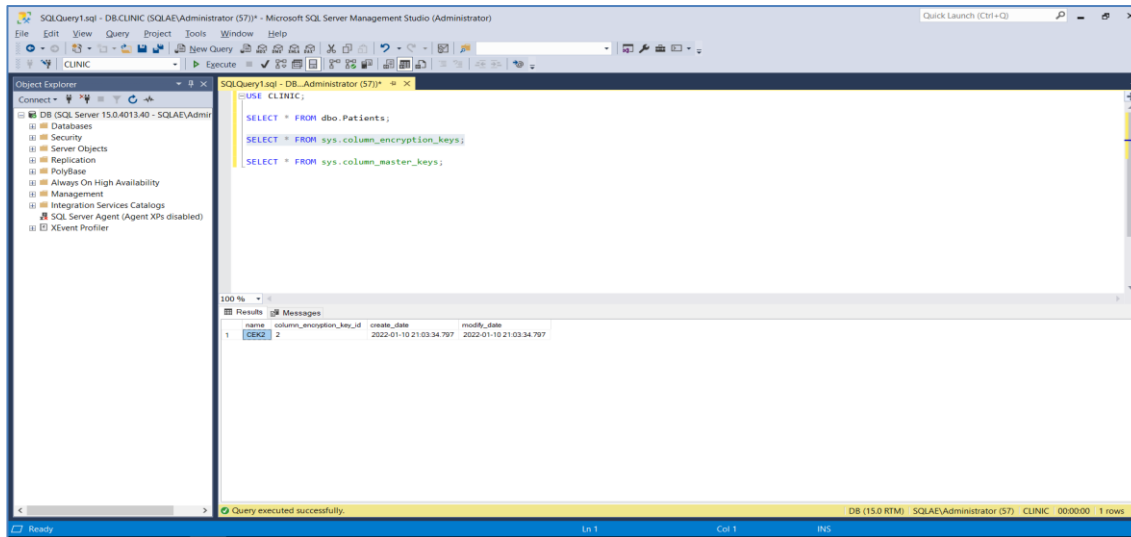
PS C:\Users\Administrator>

```

When the script gets completed successfully, it will create the new column encryption key, decrypt all encrypted columns with existing CEK and re-encrypt all columns using new CEK, store the new encrypted CEK in metadata of the database and remove the old CEK from the metadata.

3. Connect to the **DB** server and confirm the new CEK metadata created in the database.

```
SELECT * FROM sys.column_encryption_keys;
```



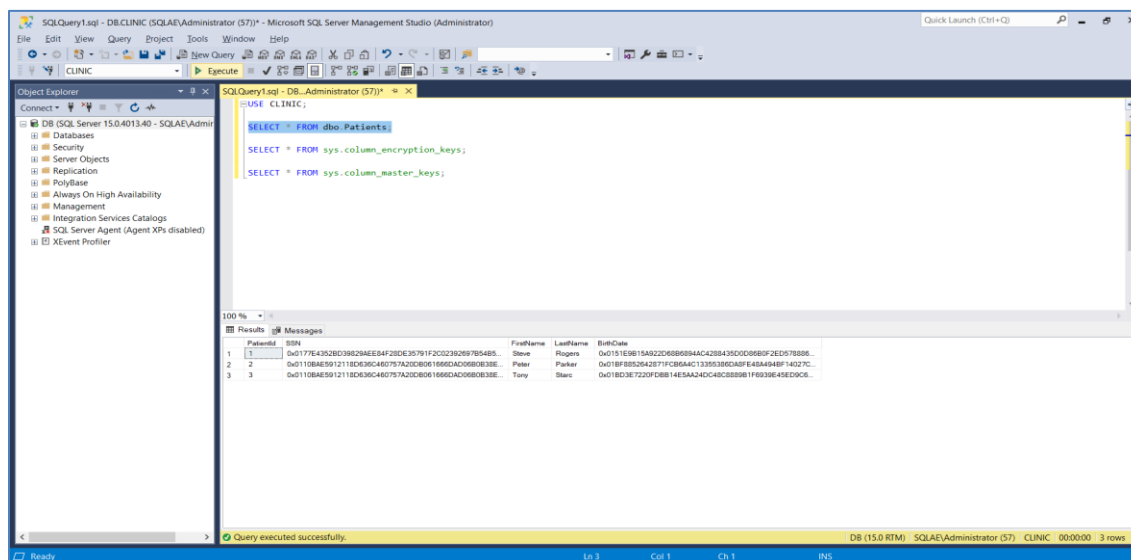
The rotation of column encryption key is completed successfully and you can view the encrypted columns (encrypted using new CEK) using [View Always Encrypted Data](#).

Remove Always Encrypted Column Encryption

If always encrypted columns need to be decrypted, we can remove column encryption from previously encrypted column data.

Example 1: Decrypting a Column

The following example shows how to decrypt a particular column in a table that is currently encrypted in a database. The image below shows the Patients table that has both SSN and BirthDate columns encrypted.



To decrypt only the SSN column:

1. Log on to any Client that has access to the CMK and run the script provided below using PowerShell.

NOTE: Replace the <server name> and <database name> with actual server name and database in your environment.

Import the SqlServer module.

```
Import-Module "SqlServer"
```

Connect to your database.

```
$serverName = "<server name>"
```

```
$databaseName = "<database name>"
```

Change the authentication method in the connection string, if needed.

```
$connStr = "Server = " + $serverName + "; Database = " +  
$databaseName + "; Integrated Security = True"
```

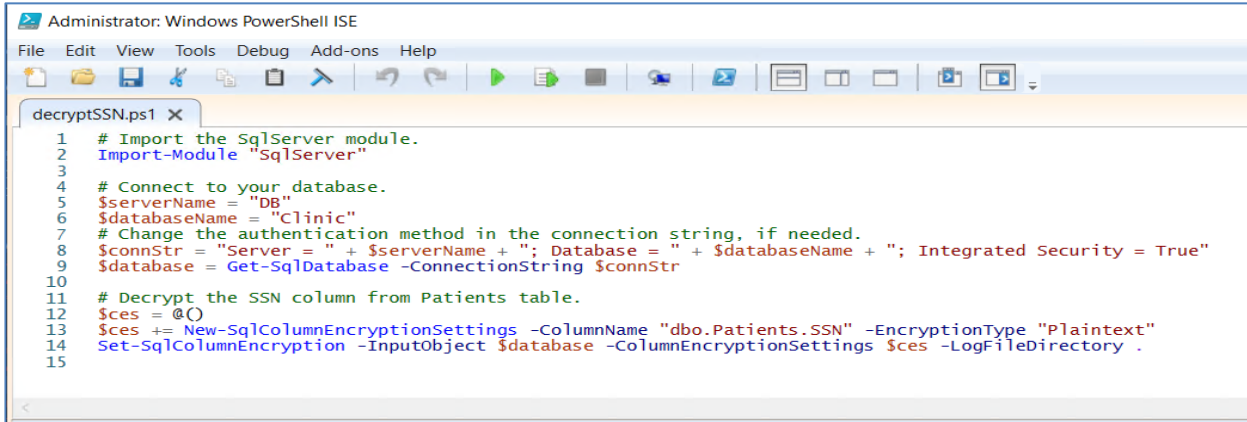
```
$database = Get-SqlDatabase -ConnectionString $connStr
```

Decrypt the SSN column from Patients table.

```
$ces = @()
```

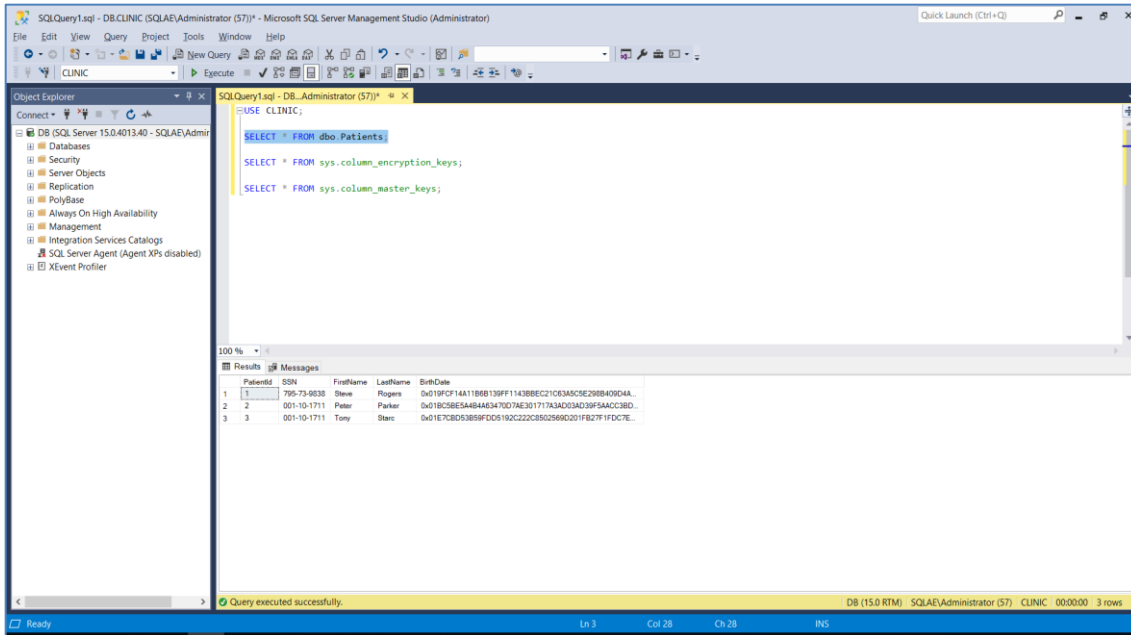
```
$ces += New-SqlColumnEncryptionSettings -ColumnName  
"dbo.Patients.SSN" -EncryptionType "Plaintext"
```

```
Set-SqlColumnEncryption -InputObject $database -  
ColumnEncryptionSettings $ces -LogFileDirectory .
```



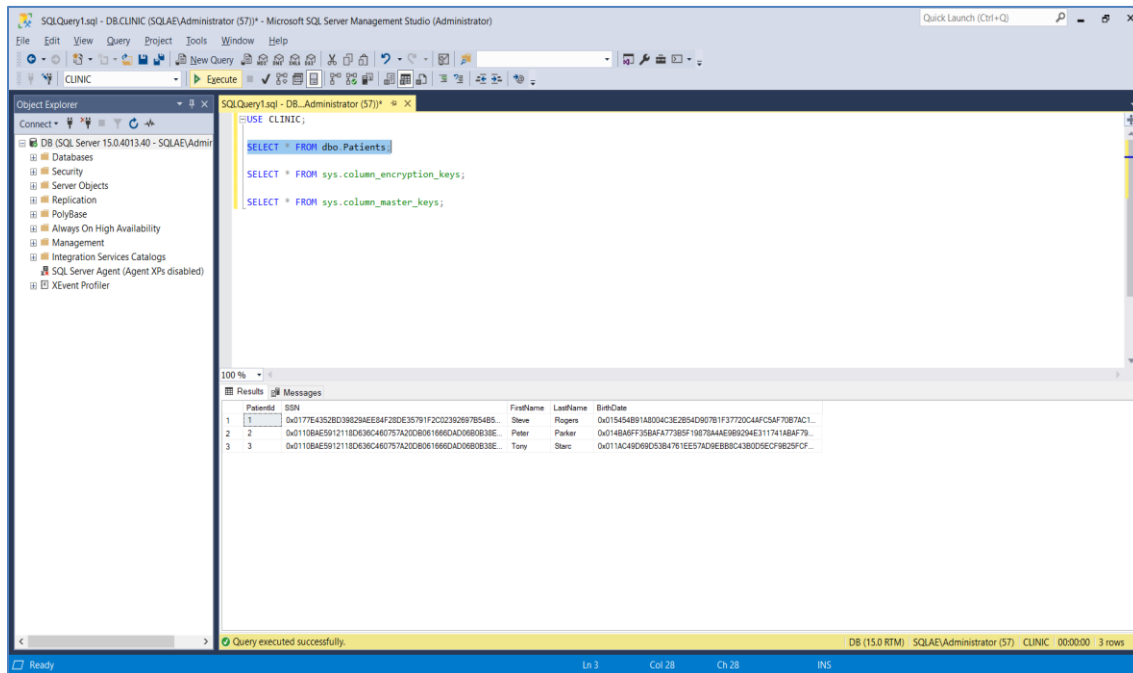
```
Administrator: Windows PowerShell ISE
File Edit View Tools Debug Add-ons Help
decrpytSSN.ps1 X
1 # Import the SqlServer module.
2 Import-Module "SqlServer"
3
4 # Connect to your database.
5 $serverName = "DB"
6 $databaseName = "Clinic"
7 # Change the authentication method in the connection string, if needed.
8 $connStr = "Server = " + $serverName + "; Database = " + $databaseName + "; Integrated Security = True"
9 $database = Get-SqlDatabase -ConnectionString $connStr
10
11 # Decrypt the SSN column from Patients table.
12 $ces = @()
13 $ces += New-SqlColumnEncryptionSettings -ColumnName "dbo.Patients.SSN" -EncryptionType "Plaintext"
14 Set-SqlColumnEncryption -InputObject $database -ColumnEncryptionSettings $ces -LogFileDirectory .
15
```

2. Check again after running the script. The SSN column will be decrypted.



Example 2: Decrypting All Columns

The next example shows how to decrypt all columns that are currently encrypted in a database. The script provided here will decrypt all columns (each and every table) in a database. The image below shows the Patients table that has both SSN and BirthDate columns encrypted.



To decrypt all the columns in a database:

1. Log on to any client who have access to the CMK and run the script provided below using PowerShell.

NOTE: Replace the <server name> and <database name> with actual server name and database in your environment.

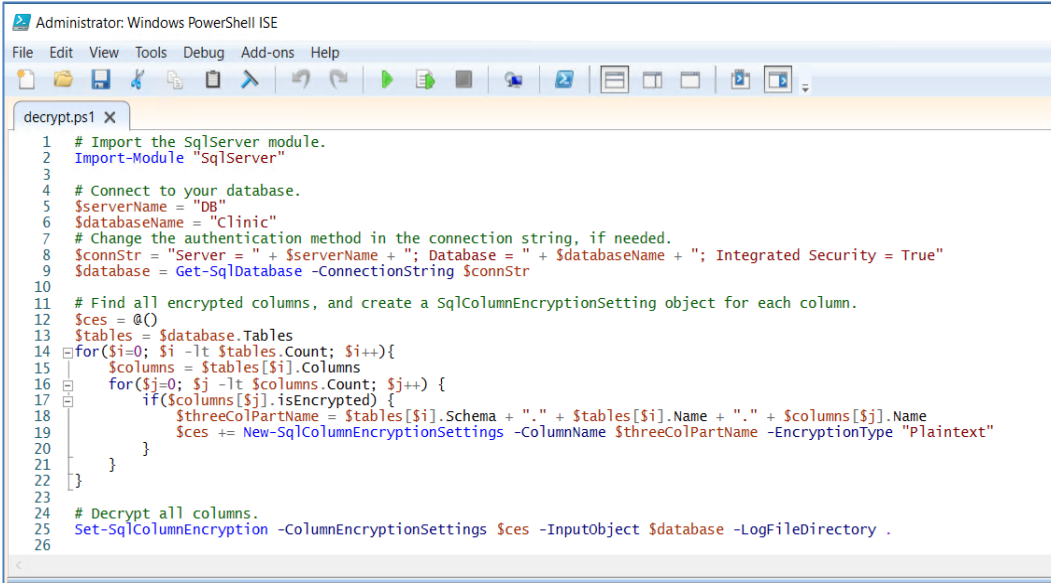
```
# Import the SqlServer module.
Import-Module "SqlServer"

# Connect to your database.
$serverName = "<server name>"
$databaseName = "<database name>"

# Change the authentication method in the connection string, if needed.
$connStr = "Server = " + $serverName + "; Database = " + $databaseName +
"; Integrated Security = True"
$database = Get-SqlDatabase -ConnectionString $connStr

# Find all encrypted columns, and create a SqlColumnEncryptionSetting object for each column.
$ces = @()
$tables = $database.Tables
for($i=0; $i -lt $tables.Count; $i++){
    $columns = $tables[$i].Columns
    for($j=0; $j -lt $columns.Count; $j++) {
        if($columns[$j].isEncrypted) {
            $threeColPartName = $tables[$i].Schema + "." +
$tables[$i].Name + "." + $columns[$j].Name
            $ces += New-SqlColumnEncryptionSettings -ColumnName
$threeColPartName -EncryptionType "Plaintext"
        }
    }
}

# Decrypt all columns.
Set-SqlColumnEncryption -ColumnEncryptionSettings $ces -InputObject
$database -LogFileDirectory .
```

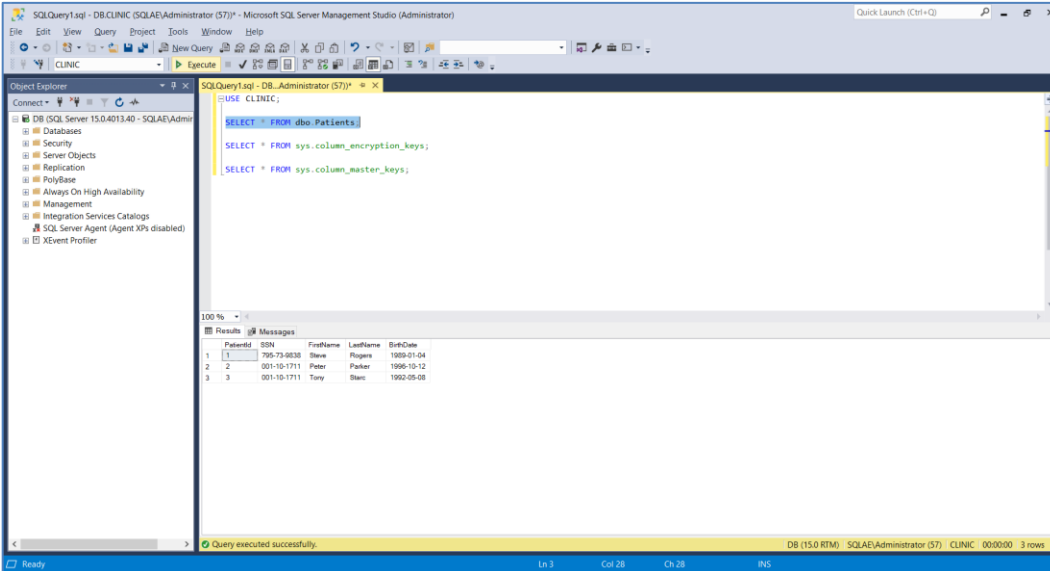


```

1 # Import the SqlServer module.
2 Import-Module "SqlServer"
3
4 # Connect to your database.
5 $serverName = "DB"
6 $databaseName = "Clinic"
7 # Change the authentication method in the connection string, if needed.
8 $connStr = "Server = " + $serverName + "; Database = " + $databaseName + "; Integrated Security = True"
9 $database = Get-SqlDatabase -ConnectionString $connStr
10
11 # Find all encrypted columns, and create a SqlColumnEncryptionSetting object for each column.
12 $ceses = @()
13 $tables = $database.Tables
14 for($i=0; $i -lt $tables.Count; $i++){
15     $columns = $tables[$i].Columns
16     for($j=0; $j -lt $columns.Count; $j++){
17         if($columns[$j].IsEncrypted) {
18             $threeColPartName = $tables[$i].Schema + "." + $tables[$i].Name + "." + $columns[$j].Name
19             $ceses += New-SqlColumnEncryptionSettings -ColumnName $threeColPartName -EncryptionType "Plaintext"
20         }
21     }
22 }
23
24 # Decrypt all columns.
25 Set-SqlColumnEncryption -ColumnEncryptionSettings $ceses -InputObject $database -LogFileDirectory .
26

```

2. Check again after running the script. All columns in the database will be decrypted.



PatientID	SSN	FirstName	LastName	BirthDate
1	795-73-0838	Irene	Parker	1989-01-04
2	001-10-1711	Peter	Parker	1996-10-12
3	001-10-1711	Tony	Blew	1992-05-08

The Always Encrypted data will revert to plaintext. If your database is protected by TDE, then the data is still being encrypted whilst at rest. Viewing the database table now will show all columns in plaintext (i.e. an unencrypted state).

NOTE: When removing Always Encryption from the database columns, ensure that all columns are appearing in plaintext. If Always Encrypted Keys are not required first delete Column Encryption Keys before dropping the Column Master Key.

This completes the integration of SQL Server Always Encrypted using Luna HSM, also covering the rotation and removing of always encrypted keys when not-in-use. When implemented, Always Encrypted, column encryption key is encrypted using column master key and the column master key will remain secured in a Luna HSM and only client, who has access to the CMK stored in the Luna HSM, will be able to view the encrypted data.

Contacting Customer Support

If you encounter a problem during this integration, contact your supplier or [Thales Customer Support](#). Thales Customer Support operates 24 hours a day, 7 days a week. Your level of access to this service is governed by the support plan arrangements made between Thales and your organization. Please consult this support plan for further information about your entitlements, including the hours when telephone support is available to you.

Customer Support Portal

The Customer Support Portal, at <https://supportportal.thalesgroup.com>, is a database where you can find solutions for most common problems. The Customer Support Portal is a comprehensive, fully searchable repository of support resources, including software and firmware downloads, release notes listing known problems and workarounds, a knowledge base, FAQs, product documentation, technical notes, and more. You can also use the portal to create and manage support cases.

NOTE: You require an account to access the Customer Support Portal. To create a new account, go to the portal and click on the **REGISTER** link.

Telephone Support

If you have an urgent problem, or cannot access the Customer Support Portal, you can contact Thales Customer Support by telephone at +1 410-931-7520. Additional local telephone support numbers are listed on the support portal.

Email Support

You can also contact technical support by email at technical.support.DIS@thalesgroup.com.