

THALES E-SECURITY

Microsoft SQL Server 2016 Always Encrypted

Integration Guide



Copyright 2017 Thales UK Limited. All rights reserved.

Copyright in this document is the property of Thales UK Limited. It is not to be reproduced, modified, adapted, published, translated in any material form (including storage in any medium by electronic means whether or not transiently or incidentally) in whole or in part nor disclosed to any third party without the prior written permission of Thales UK Limited neither shall it be used otherwise than for the purpose for which it is supplied.

Words and logos marked with ® or ™ are trademarks of Thales UK Limited or its affiliates in the EU and other countries.

Information in this document is subject to change without notice.

Thales UK Limited makes no warranty of any kind with regard to this information, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Thales UK Limited shall not be liable for errors contained herein or for incidental or consequential damages concerned with the furnishing, performance or use of this material.

Table of Contents

Table of Contents	3
Table of Figures	4
Always Encrypted and Thales nShield HSMs	5
Introduction to Always Encrypted	5
Always Encrypted Integration using Operator Cards and/or Module protection.....	5
Requirements	6
Using multiple on-premises client servers.....	6
Database Permissions	6
Security Worlds and key protection	7
Application Key Tokens.....	7
Configuring nShield Hardware Security Modules for use with Always Encrypted	8
Install and register the CNG provider.....	8
Always Encrypted using SSMS	14
Creating the Always Encrypted Column Master Key using the nCipher KSP.....	14
Enable Always Encrypted.....	19
Removing column encryption	26
Always Encrypted using PowerShell: without Role Separation.....	29
Install and Configure SqlServer PowerShell module	29
Install the Thales nCipher CNG provider	29
Creating the Always Encrypted Column Master Key using the nCipher KSP.....	30
Creating the Column Encryption Key	30
Encrypting Columns with the Column Encryption Key	31
Remove Always Encrypted Column Encryption	31
Query the encrypted columns	32
Always Encrypted using PowerShell: with Role Separation.....	33
Install and Configure SqlServer PowerShell module and verify KSP	33
Creating the Always Encrypted Column Master Key using the nCipher KSP.....	34
Encrypt Columns using SSMS	38
Encrypt Columns using PowerShell (ISE).....	38
Remove Always Encrypted Column Encryption	39
Query the encrypted columns	40
Glossary of PowerShell SqlServer CMDlets.....	41
Feature Details.....	42
Troubleshooting.....	43

Table of Figures

Figure 1: Always Encrypted example using HSM to protect CMK	7
Figure 2: Install and register nShield provider	8
Figure 3: CNG install Welcome screen.....	8
Figure 4: Select to enable / disable Pool Mode	9
Figure 5: Set Module States.....	9
Figure 6: Optional setting to enable module for remote shares.....	10
Figure 7: Present ACS card when prompted	10
Figure 8: Set Key Protection	11
Figure 9: Writing the Operator Card Set.....	12
Figure 10: Register CNG Providers	12
Figure 11: CNG provider registry Path	13
Figure 12: New Column Master Key	14
Figure 13: Generate new CMK	15
Figure 14: nCipher KSP - Create new key.....	15
Figure 15: Select module or OCS.....	16
Figure 16: Select card set by name.....	16
Figure 17: Enter passphrase.....	17
Figure 18: Card reading complete.....	17
Figure 19: New CMK generated.....	17
Figure 20: New CMK.....	18
Figure 21: Encrypt Columns	19
Figure 22: Column selection and encryption type	19
Figure 23: Select CMK to use	20
Figure 24: Run Settings	20
Figure 25: Verify Settings	21
Figure 26: Load CMK	21
Figure 27: Enter passphrase for OCS protecting the CMK.....	22
Figure 28: Confirmation of card reading	22
Figure 29: Load key.....	23
Figure 30: Enter passphrase.....	23
Figure 31: Card reading complete.....	24
Figure 32: CEK successfully encrypted column	24
Figure 33: Showing encrypted columns	25
Figure 34: Select Encrypt Columns.....	26
Figure 35: Choose the option - Plaintext	26
Figure 36: Confirm that database is off-line.....	27
Figure 37: Review column decryption state.....	27
Figure 38: Successfully removed Always Encrypted column encryption.....	28
Figure 39: SqlServer module installed	29
Figure 40: Parameterization for Always Encrypted.....	32
Figure 41: Adjust Connection Parameters	32
Figure 42: Verify Thales KSP is installed	33
Figure 43: CMK imported into Database	36
Figure 44: CEK creation.....	37
Figure 45: Permissions Status	38
Figure 46: Parameterization for Always Encrypted.....	40
Figure 47: Adjust Connection Parameters	40

Always Encrypted and Thales nShield HSMs

Introduction to Always Encrypted

Always Encrypted is a feature in Windows SQL Server 2016 designed to protect sensitive data both at rest and in flight between an on-premises client application server and Azure or SQL Server database(s).

Data protected by Always Encrypted remains in an encrypted state until it has reached the on-premises client application server, this effectively mitigates man in the middle attacks and provides assurances against unauthorized activity from rogue DBAs or admins with access to Azure / SQL server Databases. Always Encrypted was designed to be used in conjunction with **Transparent Data Encryption** however; TDE is **NOT** a requisite for implementing Always Encrypted.

Configuring Always Encrypted involves creating and provisioning cryptographic keys, specifically:

- A **Column Master Key** – The CMK, is an asymmetric RSA encryption key of size 2048 bits
- One or more **Column Encryption Key(s)** – A CEK, is a symmetric AES key of size 256 bits.

The CEK is responsible for encrypting the database column data while the CMK is protected by the HSM and is responsible for wrapping (encrypting) the CEK.

The table below shows current support for the different data operations.

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

The Column Master Key is generated using the Thales nCipher CNG provider via the HSM and the key(s) stored in an encrypted state on the on-premises client application server in the %NFAST_KMDATA%\local folder.

Note: It is recommended that the server configured with Always Encrypted be located on a different server than that on which the database resides.

Always Encrypted supports two named types of encryption, **Deterministic** and **Randomized**. Selecting deterministic encryption means that the same encrypted value will be produced from the same plaintext value each time encryption occurs, this allows for point lookups, equality joins, grouping and indexing on encrypted columns. However, this has implications on the security of the data as it potentially allows an attacker to 'guess' the plaintext from the recurring cipher text through emerging patterns within the encrypted columns. Deterministic encryption should not really be used where a small set of values are presented, e.g. 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, eliminating the predictable aspects associated with deterministic encryption, however, this also removes the ability to perform any search operations on the encrypted data in situ.

Always Encrypted Integration using Operator Cards and/or Module protection.

In order to cover both methods of protection currently available using CNG and Thales HSMs, the guide performs Always Encrypted configuration using both *Operator Card Sets* and *Module* protection for the Column Master Key (CMK). *Operator Card Set* protection (OCS are physical tokens in the form of a quorum of smart cards) is covered in the "Always Encrypted using SSMS" whereas *Module** protection is covered in the section "Always Encrypted using PowerShell".

- * Module protection utilises an AES 256 bit symmetric key with 128 bit security secured by the Security World *Module* key which is stored in the HSM hardware FIPS 140-2 level 3 boundary.

Requirements

This integration guide provides a step by step account detailing the configuration of the SQL server 2016 Always Encrypted feature. The guide covers the configuration of Always Encrypted by reference to both the Microsoft Always Encrypted wizard GUI accessed via the SSMS and Microsoft PowerShell ISE for provisioning the required cryptographic key(s). It is recommended that you read through the entirety of the guide before proceeding to implement Always Encrypted.

N.B. Always backup you database before performing any activity that may compromise database availability.

The integration was performed and tested using the following configuration:

- Microsoft Windows 2016 Hosting SQL Server 2016 Database
- Microsoft Windows 2016 for “On-Premise” client sever
- .NET Framework 4.6.1
- SQL Server Management Studio 17.x (SSMS)
- PowerShellGet 1.1.3.1
- PowerShell version 5.1 (Desktop)
- Thales nShield HSM with Security World software 12.40.xx
- Thales nShield Hardware Security module (nShield Solo +; nShield Connect +; nShield Solo XC)

You must have at least .NET Framework 4.6.1 on the on-premises client server before installing SQL Server Management Studio (SSMS). The download for .NET framework 4.6.1 can be obtained via the link below:

.NET Framework 4.6.1 download URL:

<https://www.microsoft.com/en-us/download/details.aspx?id=49982>

The Always Encrypted integration process requires administrator level access to both the “on-premises” client server and target database server for initial configuration of the Column Master Key and Column Encryption Key, thereafter Column Encryption is performed entirely via the Client Application server.

Using multiple on-premises client servers

In order for multiple on-premises client application servers to share and decrypt database columns encrypted with HSM assisted Always Encrypted, there is a requirement that:

- each client server wanting access to the contents of data encrypted with a given Column Encryption key protected by a specific Column Master Key that the server must have access to an HSM in the same Security World
- and have a copy of the Column Master Key Application key token stored on its local drive in “C:\Program Data\nCipher\Key Management Data\local” (%NFAST_LOCAL%).

By default “C:\Program Data” is a hidden folder. To view this folder open an explorer window go to the “View” tab and tick the check box named “Hidden items”.

For more information about:

- Configuring a Thales nShield HSM, see the Installation Guide for your HSM available on the DVD supplied with the HSM.
- Security World Configuration, see the appropriate User Guide for your HSM.

Database Permissions

There are four permissions required for Always Encrypted:

Operation	Description
ALTER ANY COLUMN MASTER KEY	Required to create and delete a column master key
ALTER ANY COLUMN ENCRYPTION KEY	Required to create and delete a column encryption key
VIEW ANY COLUMN MASTER KEY	Required to access and read the metadata of the column master keys to manage keys or query encrypted columns
VIEW ANY COLUMN ENCRYPTION KEY	Required to access and read the metadata of the column encryption key to manage keys or query encrypted columns

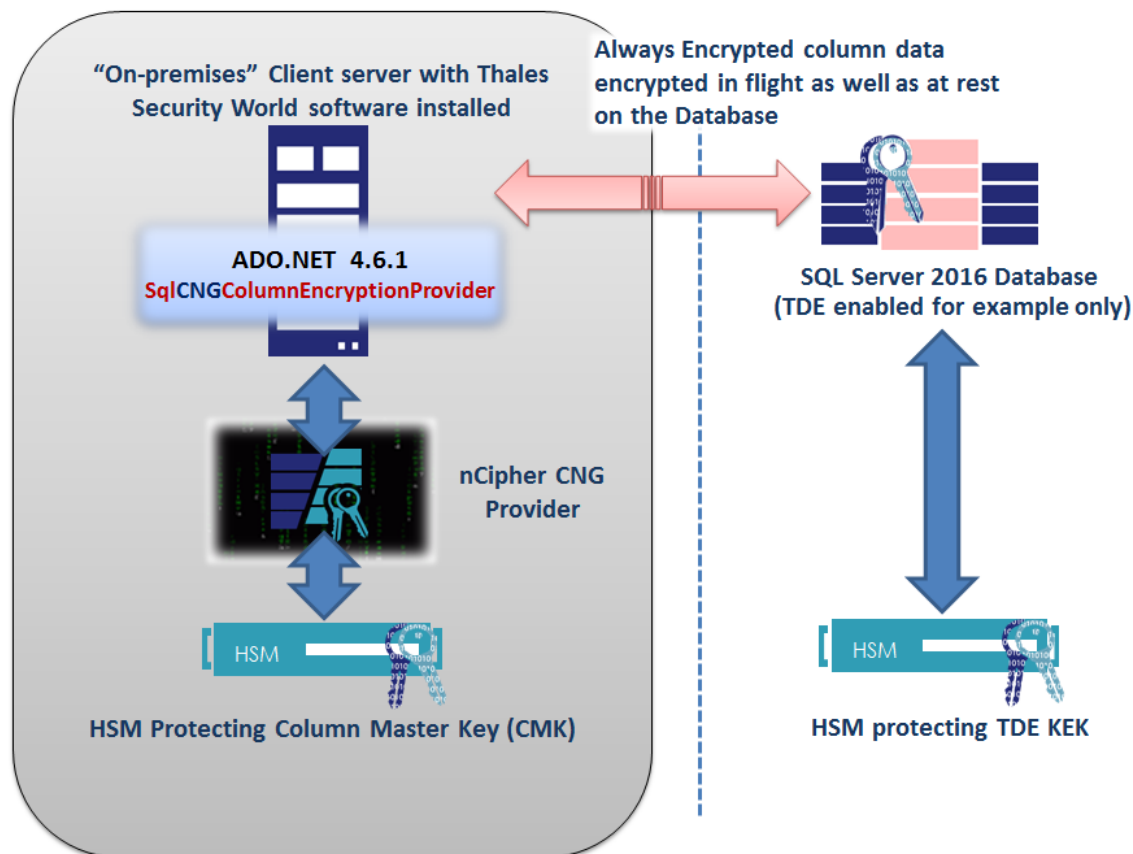


Figure 1: Always Encrypted example using HSM to protect CMK

The Thales Security World Software must be installed onto the “on-premises” client application server(s) utilizing the SQL Server 2016 Always Encrypted feature.

Note: If you are running TDE with nShield HSMs the same Security World can be used or if preferred an entirely different Security World can be implemented. If you prefer to use a different Security World you will need further HSMs as the nShield HSM can only host a single Security World instance at any one time.

Security Worlds and key protection

This section covers the options for Security World when using Always Encrypted. Always Encrypted uses the nCipher CNG provider; there are certain restrictions on the use of this provider concerning methods of authentication and operations that are available. The table below shows the restrictions on HSM key protection methods available when using the Thales nCipher CNG provider.

Security World Type	Protection type	Supported	Works in Pool mode
FIPS 140-2 level2	Module	Yes	Yes
	Softcard	No	No
	Operator Card Set 1/ n	Yes	No
	Operator Card Set k / n	Yes	No

Table 1: Supported key protection methods for nCipher CNG provider

Application Key Tokens


Application Key Tokens are an encrypted form of a Security World generated cryptographic key. These Key Tokens must not be mistaken for or regarded as being a “Key” in or of itself. The key is at all times obfuscated in this encrypted form and is only available for use as a cryptographic key when copied to the FIPS 140-2 level 3 security boundary of a correctly configured Thales Hardware Security Module.

Configuring nShield Hardware Security Modules for use with Always Encrypted

Ensure that the Thales Security World software is installed on the on-premises client server(s) utilizing the Always Encrypted feature.

Install and register the CNG provider

Once the Security World Software has been installed you must run the CNG install wizard to install and register the Thales Key Storage Provider (KSP). This can be performed via the CNG install wizard that can be found in the “Apps By name” screen of the Desktop.

Click the start button and then click on the  to access all applications. Look for the recently installed nCipher utilities.

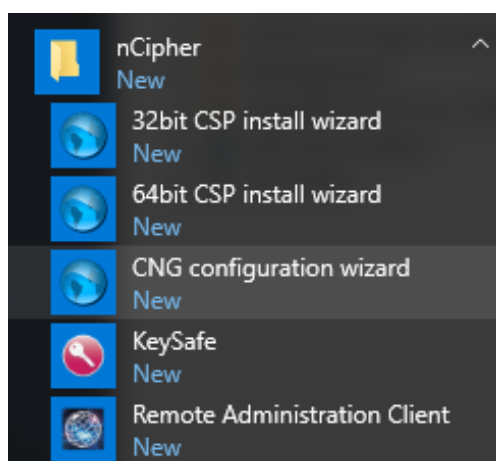


Figure 2: Install and register nShield provider

Double click the CNG configuration wizard. (If the User Access Control prompt pops up click “YES” to continue.)



Figure 3: CNG install Welcome screen

The following screen (Figure 4) prompts you to enable Pool Mode. Leave the default value with the check box unticked and click “Next”.

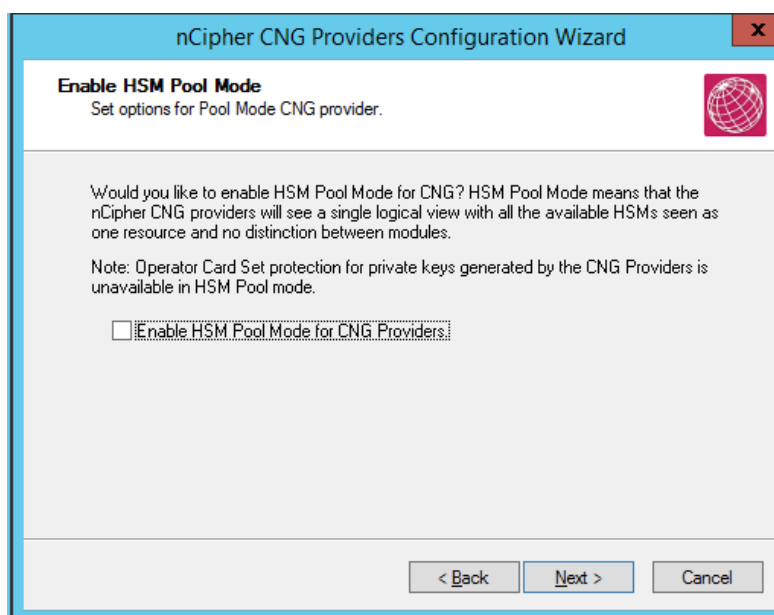


Figure 4: Select to enable / disable Pool Mode

If you already have a Security World that you intend to use for Always Encrypted the next screen will allow you to select to “**Use the existing security world**”. If you do not currently have a Security World or would like to create a new Security World then check the “**Create a new Security World**” radio button and click “Next” (for the purposes of this integration guide we have chosen to use an existing Security World).

Note: If you are creating a new Security World please refer to the Thales nShield documentation for details on creating and configuring a new Security World.

Ensure that the Set Module States show the available modules as:

- Mode = initialisation
- State = (pre)- initialisation

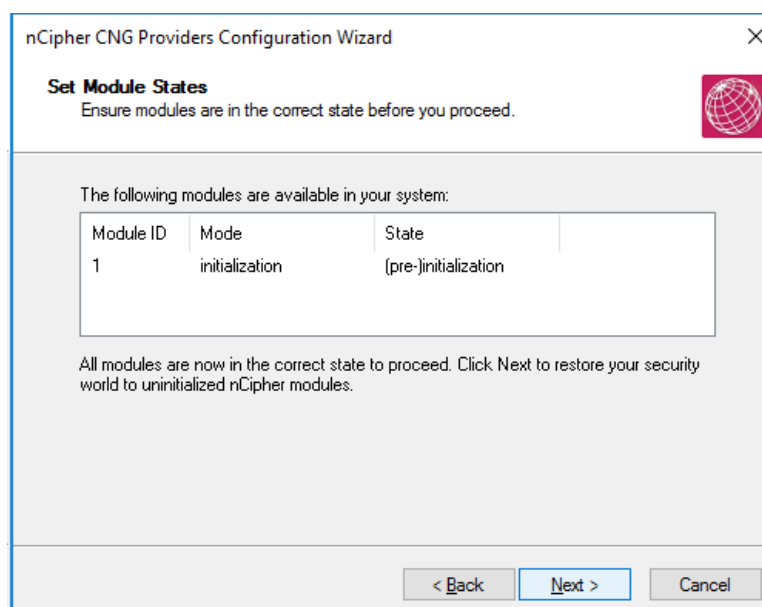


Figure 5: Set Module States

Click “Next”.

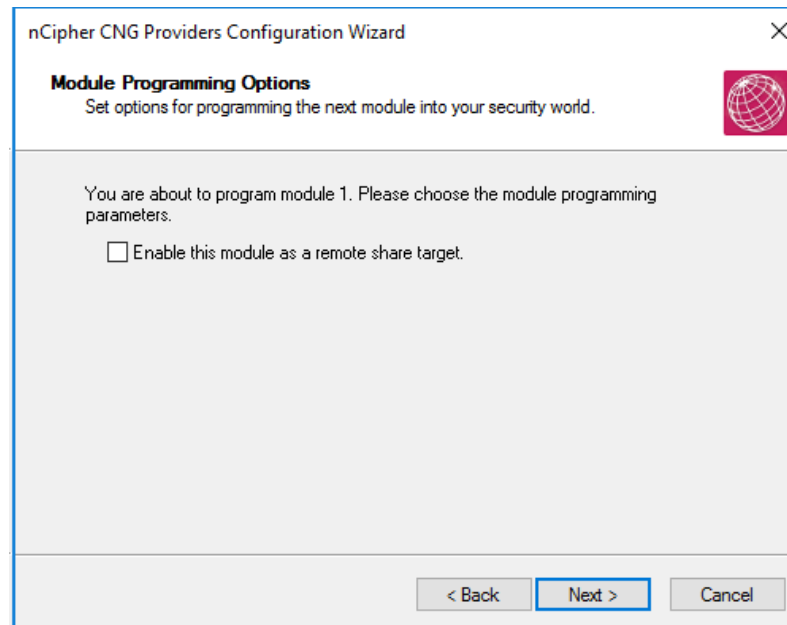


Figure 6: Optional setting to enable module for remote shares

Leave the “*Enable this module as a remote target.*” **un**-checked (Figure 6). (Please be aware that this is not to be confused with the nShield Remote Administration* utility).

Click “Next”. If you are using an existing Security World you must have the “World” file in the %NFAST_KMDATA%\local folder. Be prepared to present the quorum of Administrator cards.

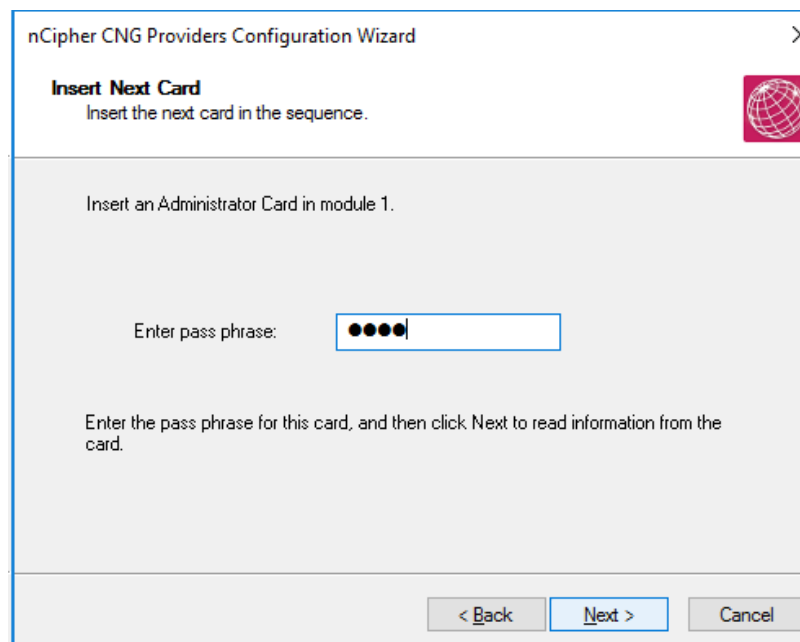


Figure 7: Present ACS card when prompted

When the ACS quorum has been presented, and the Security World loaded / created, return the HSM to “Operational” mode and choose the appropriate protection method for the Column Master Key. (Figure 8: Set Key Protection)

- * For details on Remote Administration setup and configuration please refer to the nShield Documentation on the DVD that came with your Thales HSM.

Figure 8: Set Key Protection

Proceed to create an Operator Card Set by selecting “Operator Card Set protection” and enter a name for your card set; ensure that the “Always use the wizard when creating or importing keys” is **de-selected**. Enter the card set name (this field is mandatory) then enter the required K of N value; (consult your security policy document for details on correct values to enter here). Carefully consider which of the optional values to set for the Operator Card Set. Please refer to the description in the table below for further details. Please note that by default the OCS is created as non-persistent.

Click “Next” to proceed to create the Operator Card Set.

Term	Definition
Card set name	Card set name must be supplied, unlike naming of individual cards which is optional.
Number of cards required	This relates to K of N where the value $[K]$ = the necessary number of cards required to complete authentication (the quorum) and $[N]$ = the total number of cards available. The value for K should be less than N . We do not recommend creating card sets in which K is equal to N because an error on one card would render the whole card set unusable.
Card set has a time-out	This allows a specified period of time, in seconds, where keys protected by any given OCS remain loaded in the HSM for use by your application. Once the time period has expired, all keys loaded under the OCS will be forcibly removed from the HSM such that they are no longer available. Time-outs operate independently of OCS persistence
Persistent	Keys protected by a persistent card set can be used for as long as the application that loaded the OCS remains connected to the hardware security device (unless that application removes the keys). A key protected with a persistent OCS card does not need the card to be present in the slot once the key is loaded.
Non-persistent	Keys protected by a non-persistent card set can only be used while the last required card of the quorum remains loaded in the smart card reader of the Thales hardware security device. The keys protected by this card are removed from the memory of the device as soon as the card is removed from the smart card reader.
Usable remotely	The Remote Operator feature enables the contents of a smart card inserted into the slot of one module (the attended module, such as a client module) to be securely transmitted and loaded onto another module (an unattended module, such as the nShield Connect). This is useful when you need to load an OCS-protected key onto a machine to which you do not have physical access (because, for example, it is in a secure area). This feature is deprecated in favour of Remote Administration which was launched with version 12.00 of the Thales nShield Security World software.
Recoverable PP	The option allows the recovery of a lost or forgotten pass phrase. For further details on recovery operations and Security World settings please refer to the HSM documentation supplied on the Security World media disk.

If you wish to give a name to each card, do so here, select to enter a pass phrase if required, enter and confirm the pass phrase before clicking on “Next” to create the OCS.

Note: You must have the **N** value of cards (where **N** is the total number of cards in the Set) available for this operation before you commence. Insert a card into the attached HSM card reader or the TVD (Trusted Verification Device) if you are using the Remote Administration feature, when you are prompted to do so.

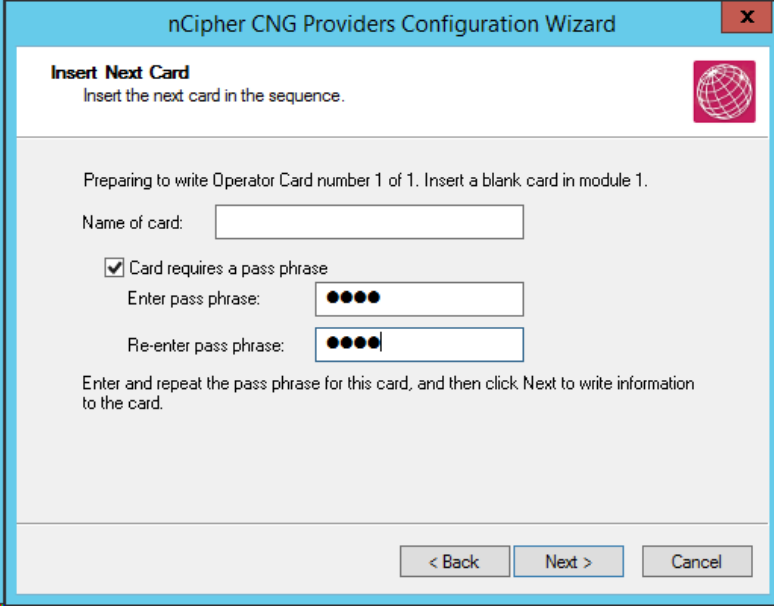
The screenshot shows the 'nCipher CNG Providers Configuration Wizard' window. The title bar is blue with the text 'nCipher CNG Providers Configuration Wizard' and a close button. The main window has a light blue header with the title 'Insert Next Card' and a subtitle 'Insert the next card in the sequence.' Below this, the text reads: 'Preparing to write Operator Card number 1 of 1. Insert a blank card in module 1.' There is a text box for 'Name of card:'. Below that is a checkbox labeled 'Card requires a pass phrase' which is checked. Under the checkbox are two text boxes: 'Enter pass phrase:' and 'Re-enter pass phrase:', both containing masked characters (dots). At the bottom, there is a note: 'Enter and repeat the pass phrase for this card, and then click Next to write information to the card.' At the very bottom are three buttons: '< Back', 'Next >', and 'Cancel'.

Figure 9: Writing the Operator Card Set

You do not have to give individual cards names, but if you wish, enter the name of the card in the appropriate field. Similarly, you do not have to give the cards a pass phase, but enter one if appropriate for your security policy. Click “Next”.

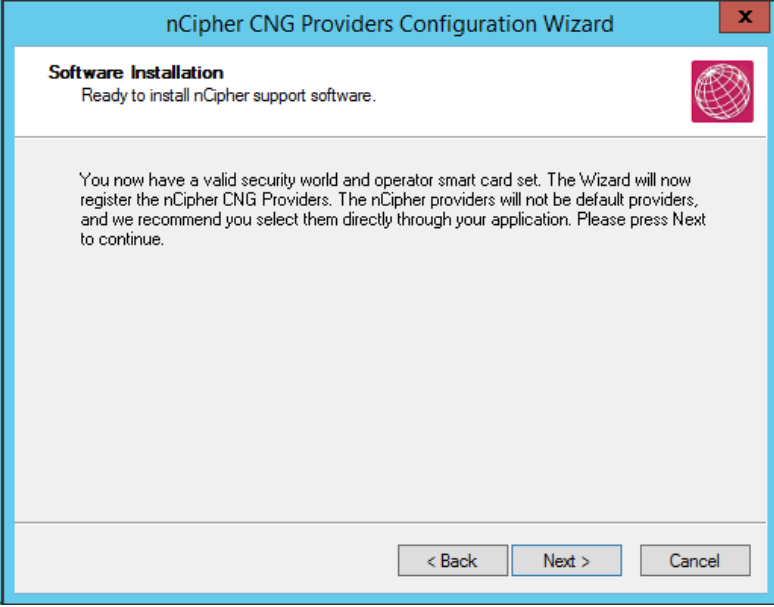
The screenshot shows the 'nCipher CNG Providers Configuration Wizard' window. The title bar is blue with the text 'nCipher CNG Providers Configuration Wizard' and a close button. The main window has a light blue header with the title 'Software Installation' and a subtitle 'Ready to install nCipher support software.' Below this, the text reads: 'You now have a valid security world and operator smart card set. The Wizard will now register the nCipher CNG Providers. The nCipher providers will not be default providers, and we recommend you select them directly through your application. Please press Next to continue.' At the bottom are three buttons: '< Back', 'Next >', and 'Cancel'.

Figure 10: Register CNG Providers

Note: If you are using Remote Administration you may need to select the **<Back** button and then return via **Next >** to the ESN confirmation screen. Re-select “OK” on the TVD to continue creating the OCS.

The Thales nCipher CNG providers will now be installed and the key Storage Provider will be registered. To confirm that the KSP has been successfully registered open either a Command Line Interface or PowerShell (right click and “Run as Administrator”) and run the following command:

```
>cnglist.exe --list-providers
```

```
PS C:\WINDOWS\system32> cnglist.exe --list-providers
Microsoft Key Protection Provider
Microsoft Passport Key Storage Provider
Microsoft Platform Crypto Provider
Microsoft Primitive Provider
Microsoft Smart Card Key Storage Provider
Microsoft Software Key Storage Provider
Microsoft SSL Protocol Provider
Windows Client Key Protection Provider
nCipher Primitive Provider
nCipher Security world Key Storage Provider
PS C:\WINDOWS\system32>
```

You should see the “nCipher Security World key Storage Provider” listed (Circled in red, above). You will find the provider in the registry at this location:

HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Control\Cryptography\Providers\ncipherSecurityWorldKeyStorageProvider

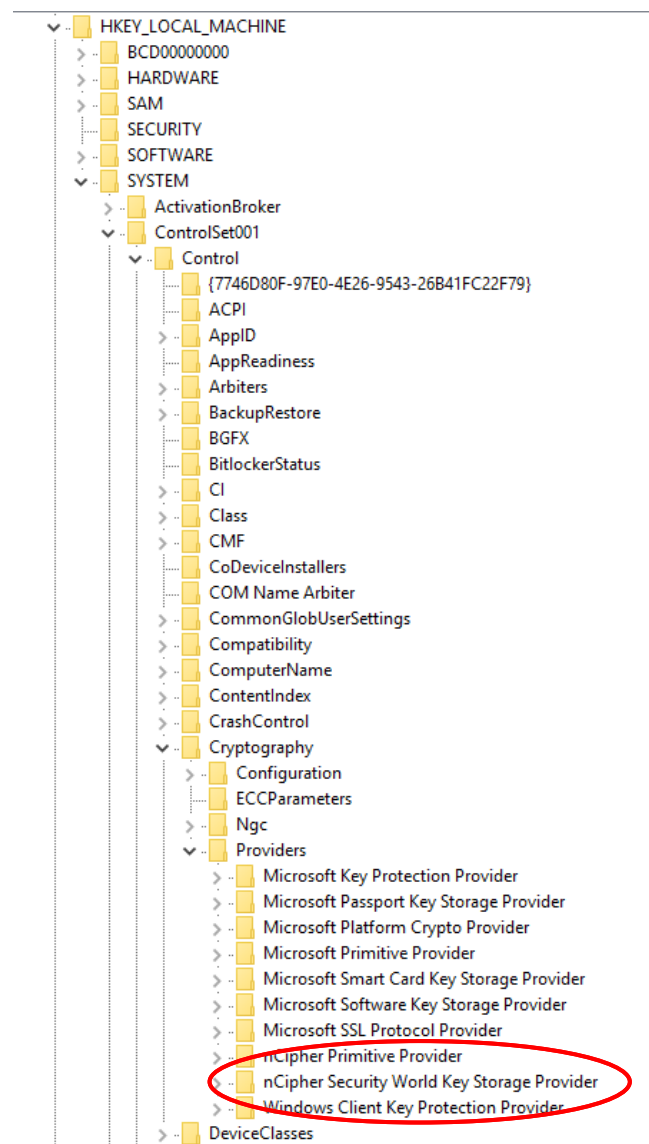


Figure 11: CNG provider registry Path

Always Encrypted using SSMS

Creating the Always Encrypted Column Master Key using the nCipher KSP

Once you have successfully installed the nCipher CNG Key Storage Provider you can begin to configure Always Encrypted.

From the “Apps by name” desktop environment, select the Microsoft SQL Server Management Studio and connect to the desired database. Once connected to the database the first thing you will need to do is create a Column Master Key. This key will encrypt all subsequent Column Encryption keys (CEKs).

Using Object Explorer, select the Security directory under the desired Database (In the example below this can be seen as “TestDatabase”). Click to expand “Always Encrypted Keys”.

Select: <Your_database> > Security > Always Encrypted Keys > Column Master Keys. Right click on “Column Master Keys” and select > New Column Master Key... the “New Column Master Key” dialogue box will open (Figure 13).

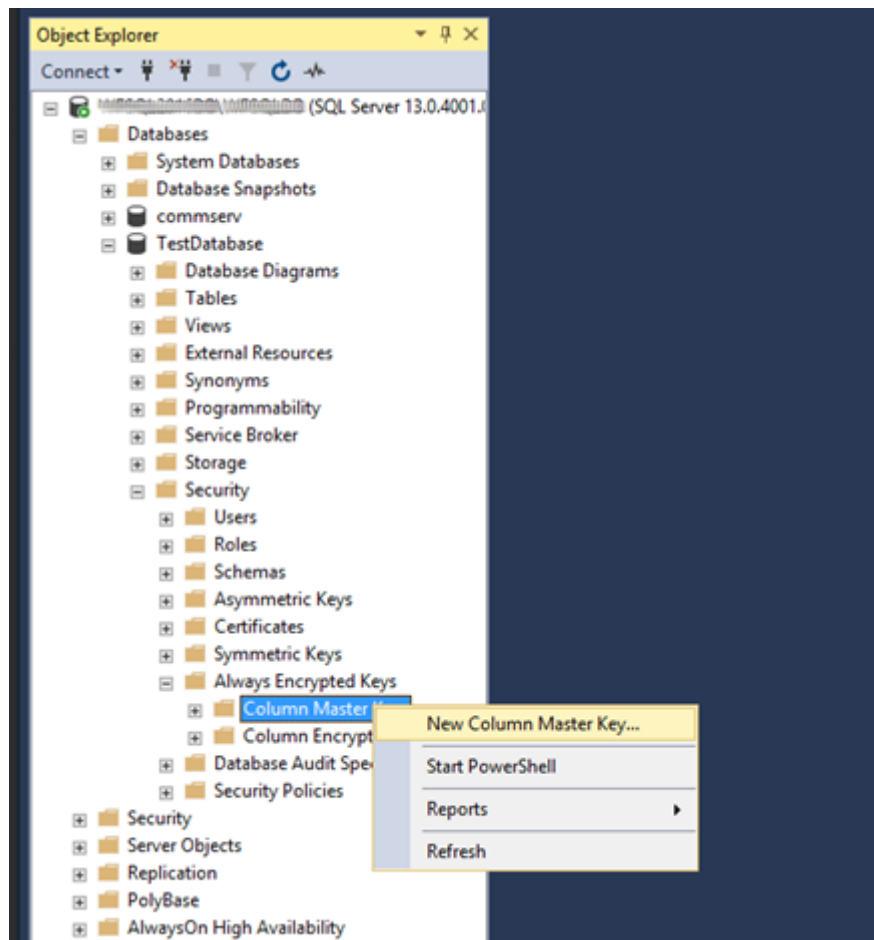


Figure 12: New Column Master Key

In the **Name** field, enter a meaningful name for the CMK, e.g. MyCMK.

From the drop down list select the “Key Storage Provider (CNG)” option. This will then present the option to “**Select a provider**”. Choose the “nCipher Security World Key Storage Provider” from the drop down list and click Generate Key to create a new CMK using the nShield HSM and CNG KSP.

If the “nCipher Security World Key Storage Provider” is not visible you will need to ensure that you have correctly installed and registered the Thales Key Storage Provider.

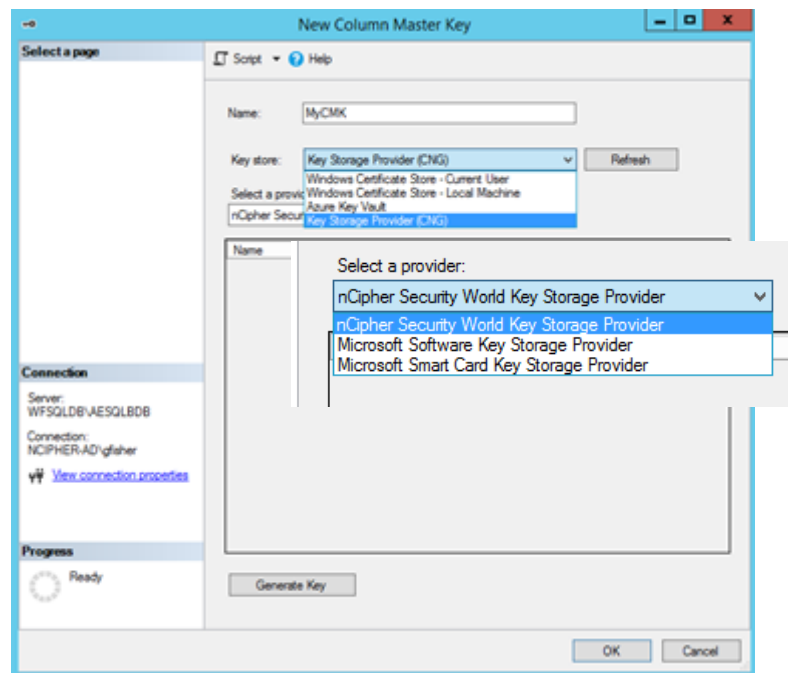


Figure 13: Generate new CMK

The “nCipher Key Storage Provider – Create key” dialogue will open.

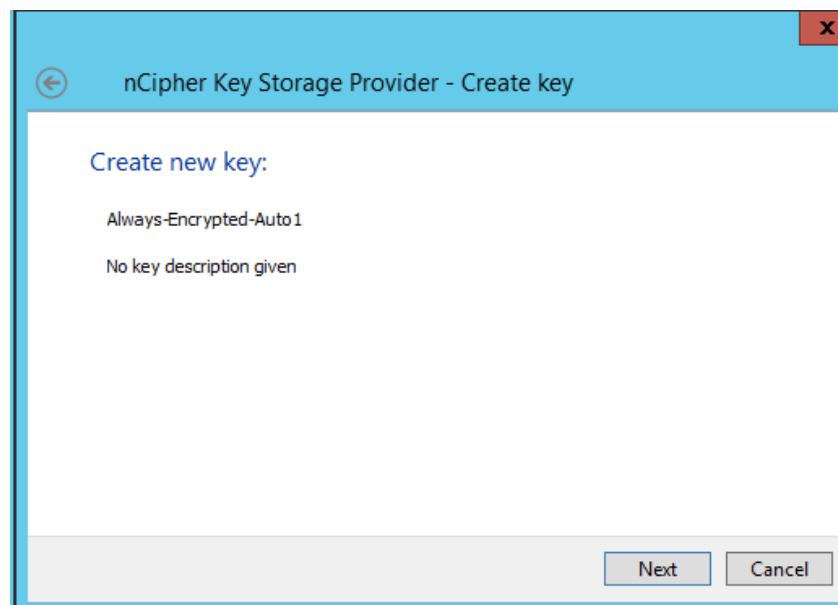


Figure 14: nCipher KSP - Create new key

Click “Next” to select key protection options.

The “Select a method to protect new key” dialogue box will open allowing you to select the appropriate method for your security policy. Select to use either Module or Operator Card Set protection. (In this example we are using Operator Card Set Protection).

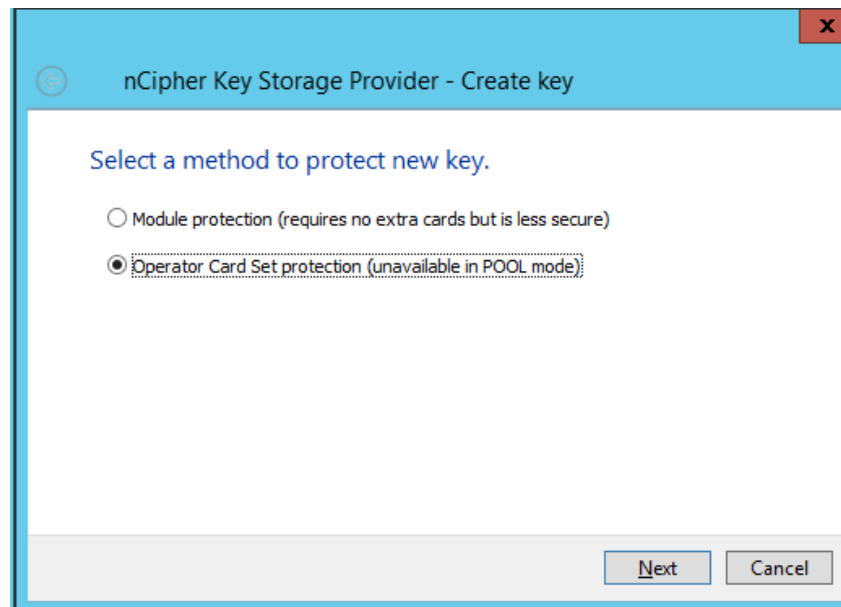


Figure 15: Select module or OCS

The following screen will prompt you to select which Operator Card Set to use for the CMK. (If you have multiple OCS all currently available Operator Card sets will be listed) Operator Card sets will be listed in the left hand field (Figure 16).

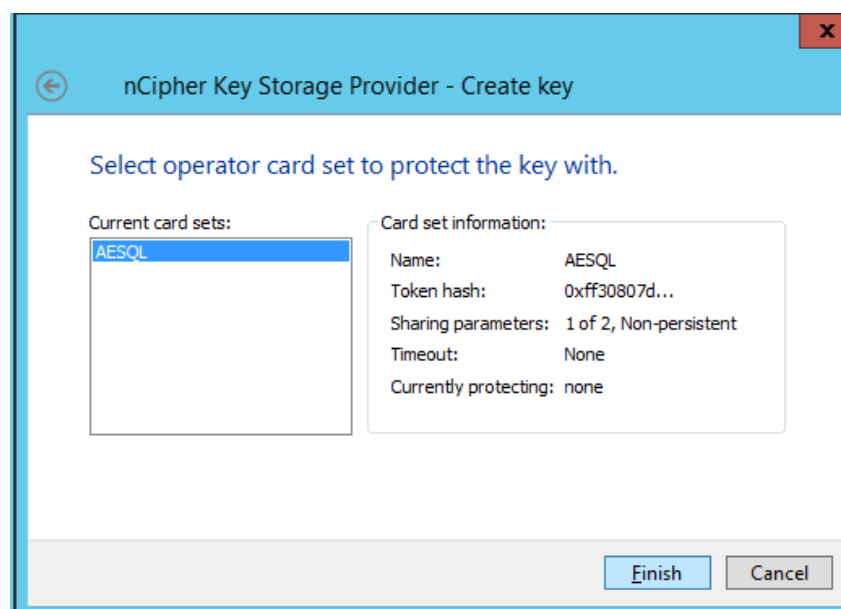


Figure 16: Select card set by name

Select the OCS that you want to use and click “Finish”. The next two screens will prompt you to enter the passphrase for the selected OCS, if one exists, and confirm that card reading completed successfully. (Ensure that you have the correct OCS available)

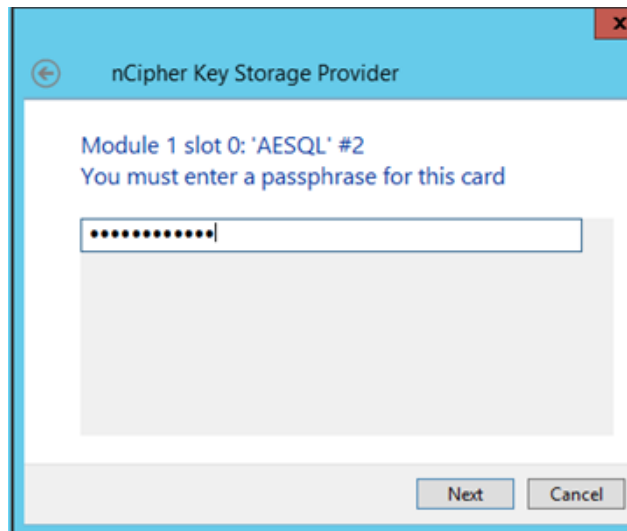


Figure 17: Enter passphrase

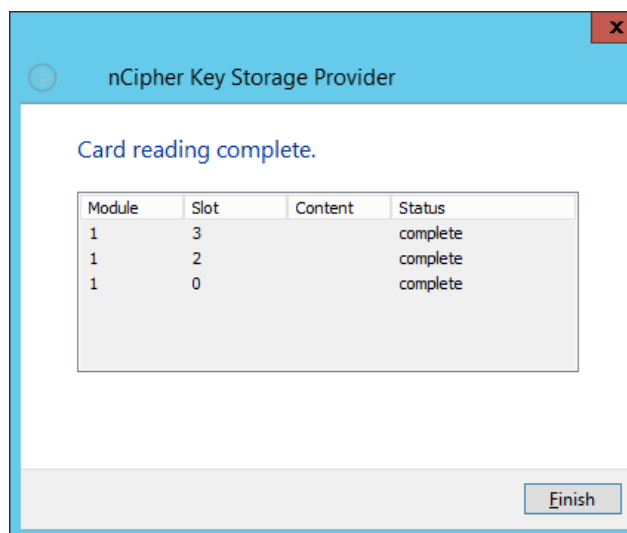


Figure 18: Card reading complete

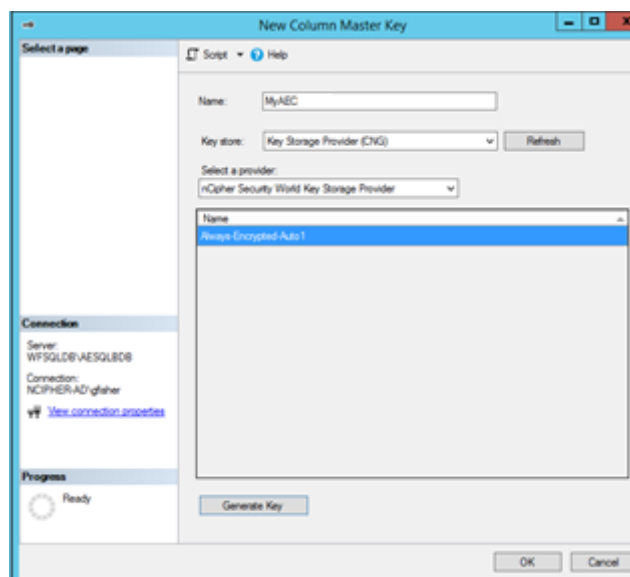


Figure 19: New CMK generated

You will now have a Column Master Key called MyCMK protected by the card set, AESQL. The newly generated CMK will be visible in the Name field.

To confirm the key has been successfully created using the Thales nShield Key Storage Provider open either a CLI or PowerShell. (This must be done with elevated permissions; right click and select “Run as Administrator”). Run the Thales utility `nfkminfo.exe` with the `-k` argument. You should see something similar to the output, seen below.

```
nfkminfo.exe -k
Key list - 1 keys
  AppName caping          Ident s-1-5-21-1277476411-3880915791-1682396242-1002--
  7cbbd9d5477b6d2ed4b6df83e3fa50ac3745b855
```

For further information about the key, including its name, and protection (i.e.. Module or Operator Card Set) run `>nfkminfo` with the `<AppName>` and `<Ident>` as reported by `>nfkminfo -k`, above.

Example Key information:

```
nfkminfo.exe -k caping s-1-5-21-1277476411-3880915791-1682396242-1002--
7cbbd9d5477b6d2ed4b6df83e3fa50ac3745b855

Key AppName caping Ident s-1-5-21-1277476411-3880915791-1682396242-1002--
7cbbd9d5477b6d2ed4b6df83e3fa50ac3745b855
  BlobKA length          1052
  BlobPubKA length        444
  BlobRecoveryKA length  1464
  name                    "MyCMK"
  hash                    94f0bd3bcf6cc1f07a06086bb7918961c67747f0
  recovery                Enabled
  protection              CardSet
  other flags              PublicKey !SEAppKey !NVMemBlob +0x0
  gentime                 2017-09-14 12:59:12
  SEE integrity key       NONE
```

Click OK, the database now has a Column Master Key protected by the Security World under OCS protection.

To view the new Column Master Key use the SQL Object Explorer. Navigate to the relevant database and expand by clicking the + sign. Expand the “Security” folder and then expand the “Always Encrypted Keys” Folder. You will find two folders, one for the Column Master Key(s) and one for the Column Encryption Key(s).

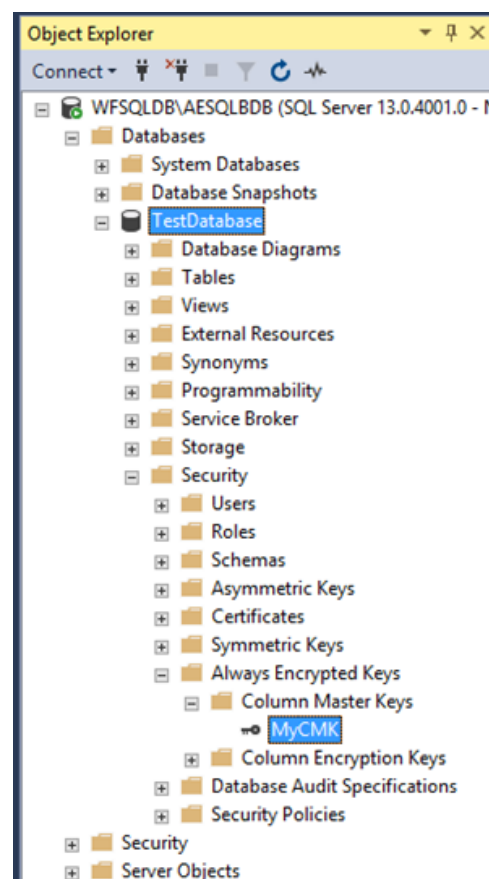


Figure 20: New CMK

Enable Always Encrypted.

To Enable Always Encrypted and generate a Column Encryption Key, right click on the required database, in this example we shall use TestDatabase, right click and in the “Tasks” tab select to “Encrypt Columns...” this will open the Always Encrypted wizard.

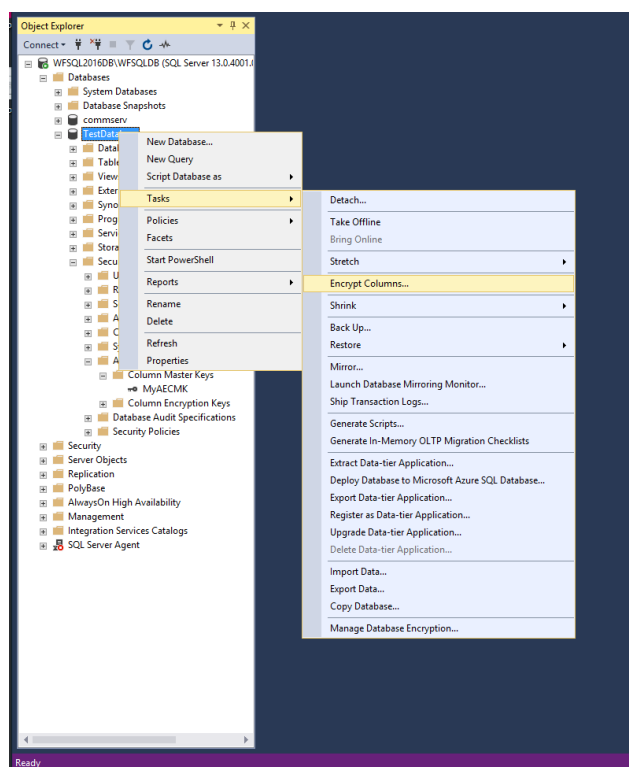


Figure 21: Encrypt Columns

If you don't want the Introduction screen presented each time you run the wizard, check the “Do not show this page again” box. Click “Next”.

The Column Selection screen allows you to choose the type of Column Encryption Key and specify the columns you want to encrypt.

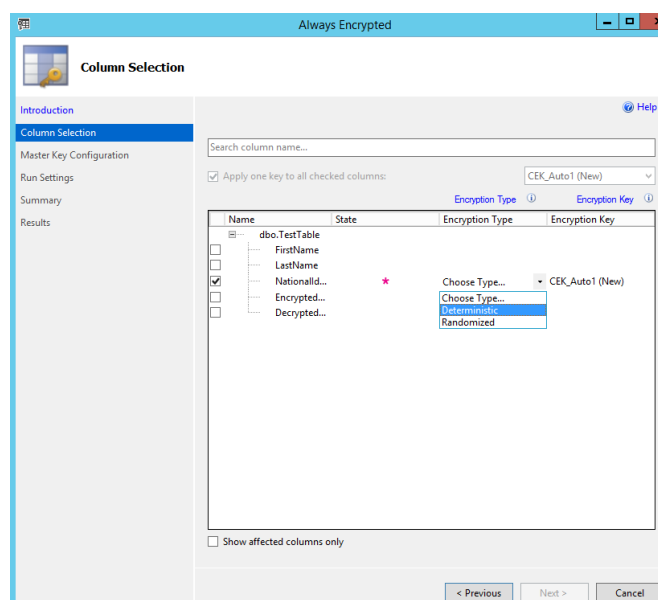


Figure 22: Column selection and encryption type

Note: The “Apply one key to all checked columns” is shaded out until such time as you have two or more CEKs available. You will then also have the option to select the CEK for any given column via the drop down list beneath the “Encryption Key” option.

Under “Encryption Type” click to select the column(s) to encrypt by checking the appropriate box to the left of the column name, you can then select the encryption method from the drop down box beneath “Choose Type” Encryption is either:

- Deterministic
- Randomized
- Plaintext (only available to revert encrypted columns to an unencrypted state)

Click “Next”.

On the Master key Configuration page, Make sure that you select the CMK that was generated using the nCipher Key Storage Provider and protected by the HSM and click next.

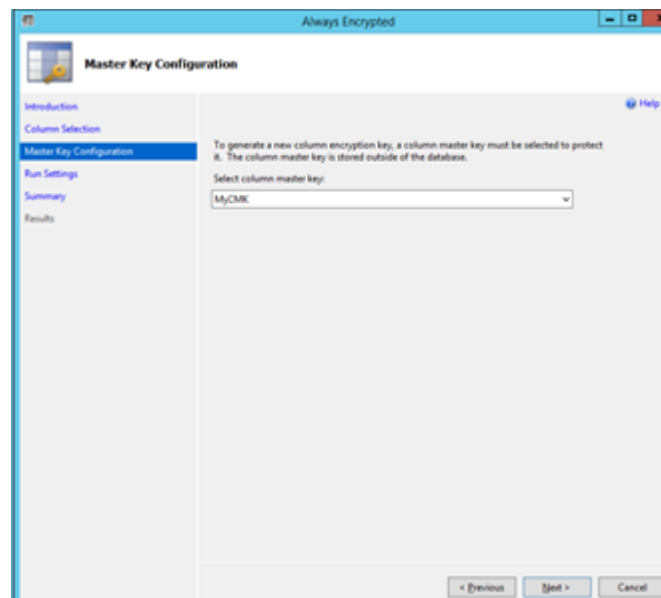


Figure 23: Select CMK to use

The process of encrypting your database records can take a considerable amount of time, depending on the size / 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.

In this case we will continue and run the encryption straight away. Select the radio button, “Proceed to finish now” this will begin the process of creating the CEK and using it to encrypt the specified column in the database. Click “Next” to view the Summary page.

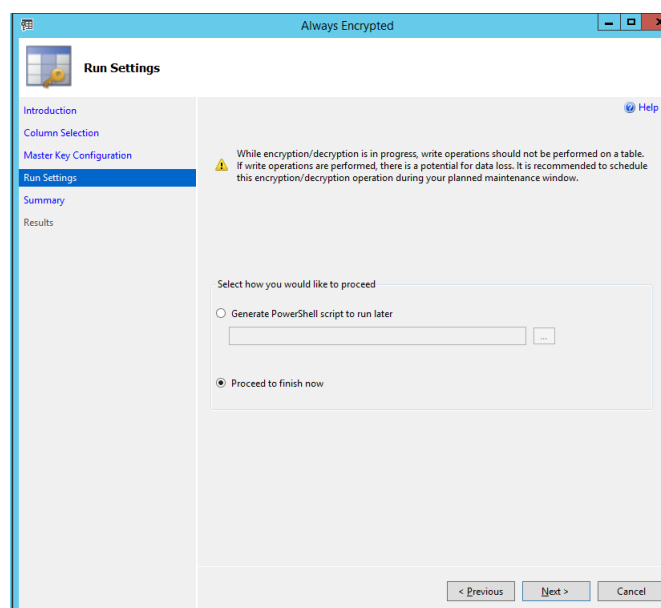


Figure 24: Run Settings

N.B. **Run Settings:** It is recommended that maintenance downtime be scheduled for this activity.

This page allows you to verify your configuration choices and amend if necessary.

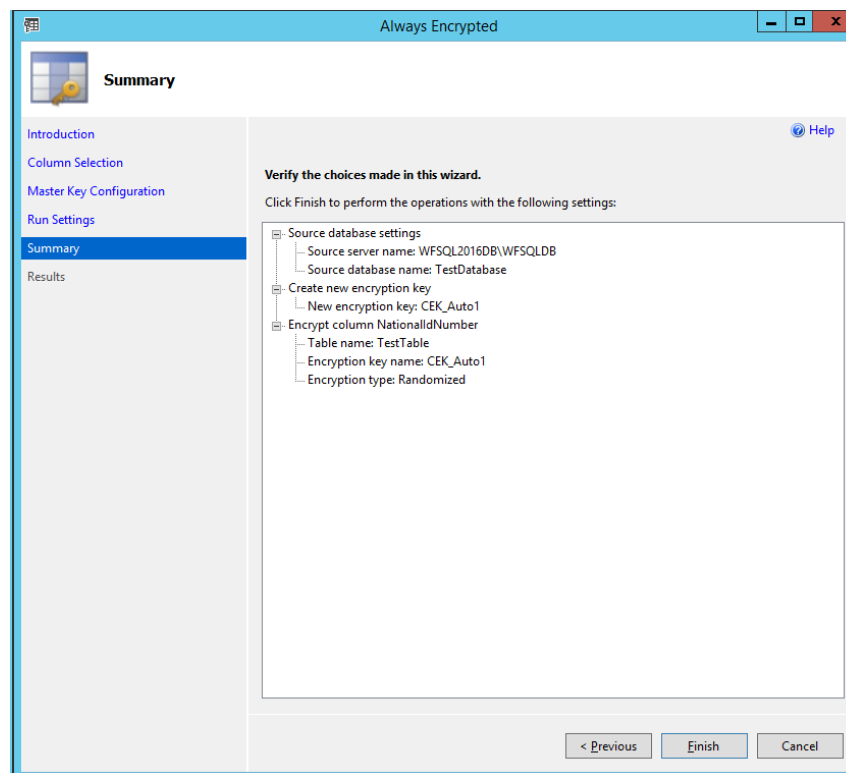


Figure 25: Verify Settings

The next operation requires the Operator Card Set quorum to be available.

Before you can create a CEK you must first load the CMK. The following screen (Figure 26) will prompt you to present the OCS protecting the Column Master Key. Present the OCS quorum and enter the passphrase, continue by clicking "Finish".

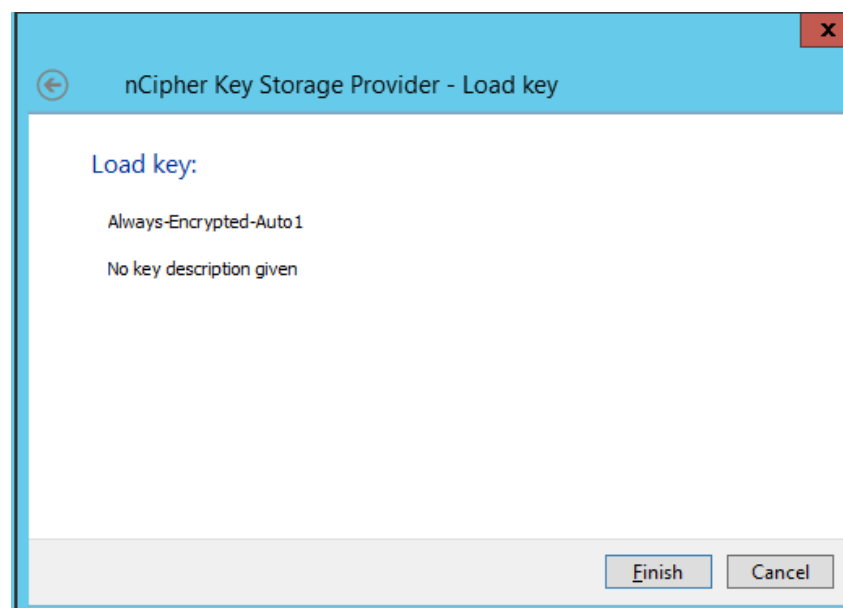


Figure 26: Load CMK

You will be prompted for the Operator Card passphrase, enter the passphrase and click “Next”.

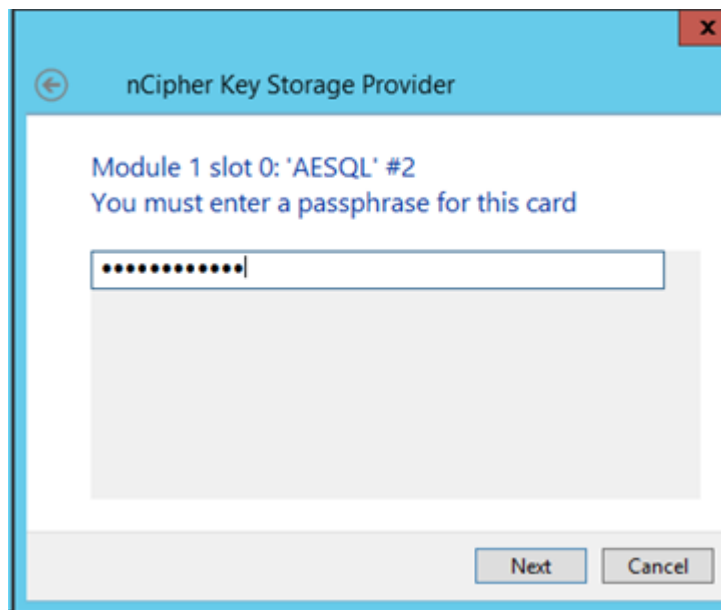


Figure 27: Enter passphrase for OCS protecting the CMK

Click “Finish” to complete the loading of the CMK into the memory of the HSM this will allow it to securely encrypt the Column Encryption Key.

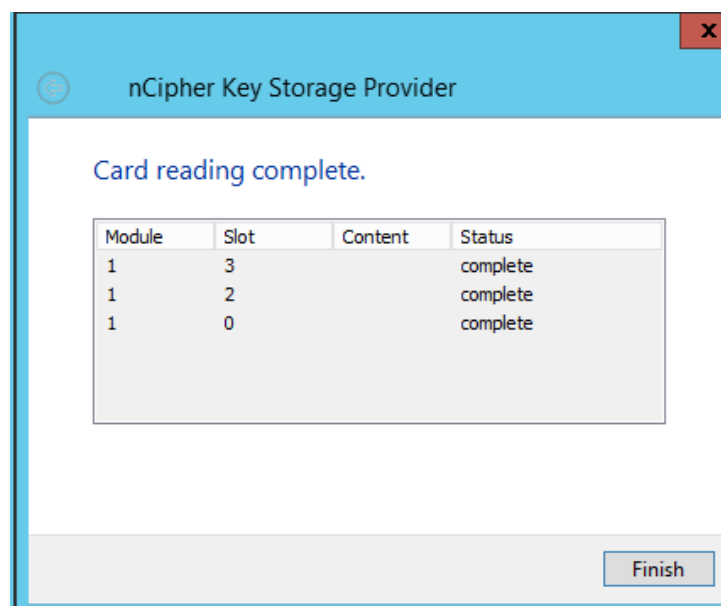


Figure 28: Confirmation of card reading

Next, the CEK shall be generated and protected by an OCS protected Column Master Key.
Click “Finish” to proceed.

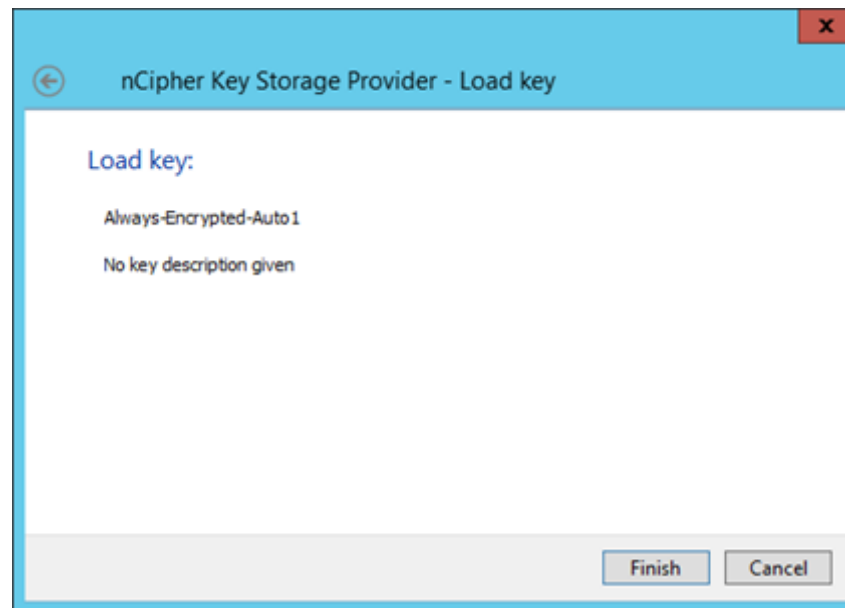


Figure 29: Load key

Insert the quorum from the Operator Card set and enter the passphrase(s) when prompted.

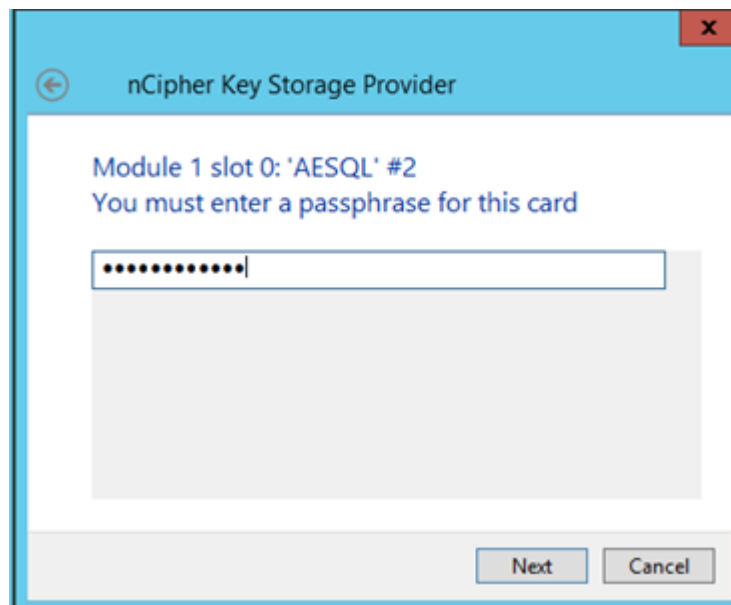


Figure 30: Enter passphrase

The following screen reports on the status of the Operator Card reading operation.

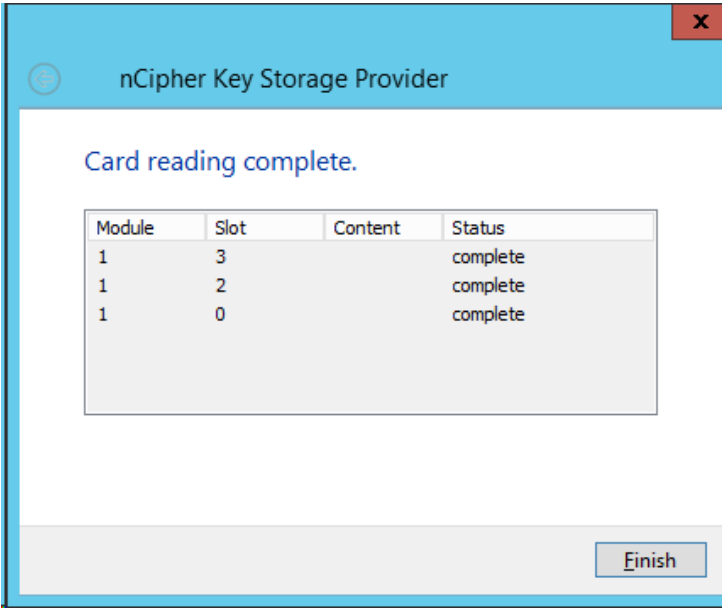


Figure 31: Card reading complete

Providing the Operator Card(s) where correctly read the CEK will have been created.

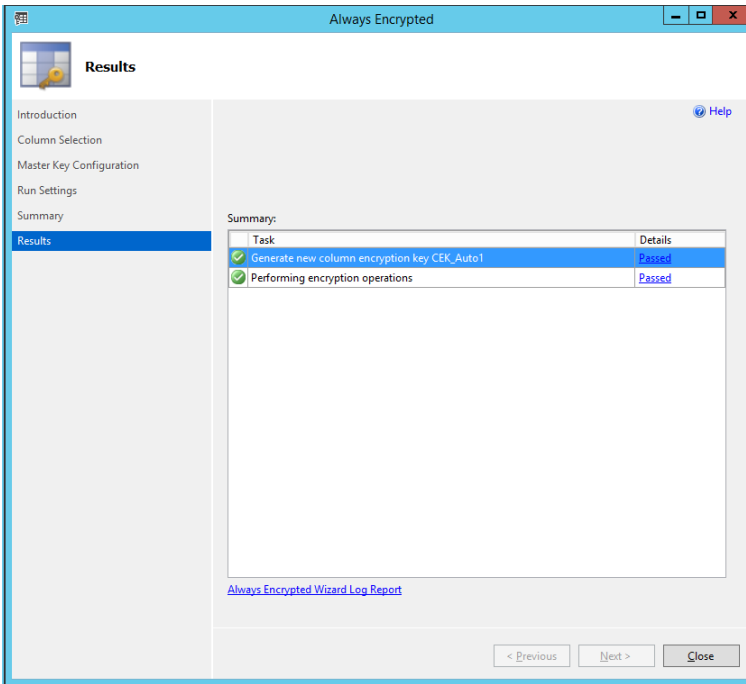


Figure 32: CEK successfully encrypted column

The Results page will report that the “CEK was generated and the requested / specified columns are now encrypted. You can now click “Close” to exit the Always Encrypted Column Encryption Key wizard.

The screenshot displays the SQL Server Enterprise Manager interface. The central pane shows a query window with the following SQL code:

```

--***** script for select top rows from command from SMS *****
SELECT TOP (1000) [FirstName]
               [LastName]
               [NationalIDNumber]
               [EncryptedNationalIDNumber]
               [DecryptedNationalIDNumber]
FROM [TestDatabase].[dbo].[TestTable]

```

The Results pane below the query window shows the execution results. The table has the following columns: First Name, Last Name, NationalIDNumber, EncryptedNationalIDNumber, and DecryptedNationalIDNumber. The row for 'Jack Shephard' is highlighted, showing a NULL value for EncryptedNationalIDNumber and a decrypted value for DecryptedNationalIDNumber.

First Name	Last Name	NationalIDNumber	EncryptedNationalIDNumber	DecryptedNationalIDNumber
1	Jack	Shephard	0x01F8B0A2A2EED052B83E754C34E360F0D52715E7A4B00	NULL
2	John	Locke	0x01C43CB015848E9C0C3C8024F8484583C8B52743429E	NULL
3	Kate	Austin	0x01433A30C1CE3A3F8C8493968009FC0F9EBE81A16E6B	NULL
4	James	Ford	0x018A1ED9A30F48708A7595A5D07C28E277E70363008	NULL
5	Ben	Lin	0x011EDC6A860093C23A7902295C7258545479A9C300D	NULL
6	Desmond	Hume	0x013ACF195BA9708C63E6A07528A4F1D6A5A0700C18C23	NULL
7	Daniel	Farley	0x0171845B462AD01853E9073B4265507844E3B43CA86F2	NULL
8	Sayid	Jarrah	0x01348E3C8B9948D0188CB49EC07770CDAC2A25A208A8E	NULL
9	Richard	Alpert	0x01B915F487D10C27C2F5C8E59526728C13F7A70DA778	NULL
10	Jacob	Smith	0x01A4C813C78603574F36878EECD38BFD12A407C0C	NULL

The Properties pane on the right shows the current connection parameters for the connection 'WFSQLEDB\AESQLEDB'.

Figure 33: Showing encrypted columns

To show the encrypted columns in plaintext (i.e. Decrypted) you should disconnect from the database and reconnect with the given additional connection parameter. This is entered from the “Connect to Database Engine” logon screen. Select the required server name and click on “Options>>” Go to “additional Connection Parameters” and add the connection string “Column Encryption Setting = enabled” (without parenthesis “”) and then click “Connect”.

When you now run the query on the table you will now see the original values decrypted by the Column Encryption Key.

Removing column encryption

If you want to remove the protection provided by Always Encrypted column encryption this can be done using the SQL Server Management Studio Object Explorer.

To remove Column Encryption from a specific or multiple data column(s):

Right click on the required database and in the “Tasks” menu select “Encrypt Columns”.

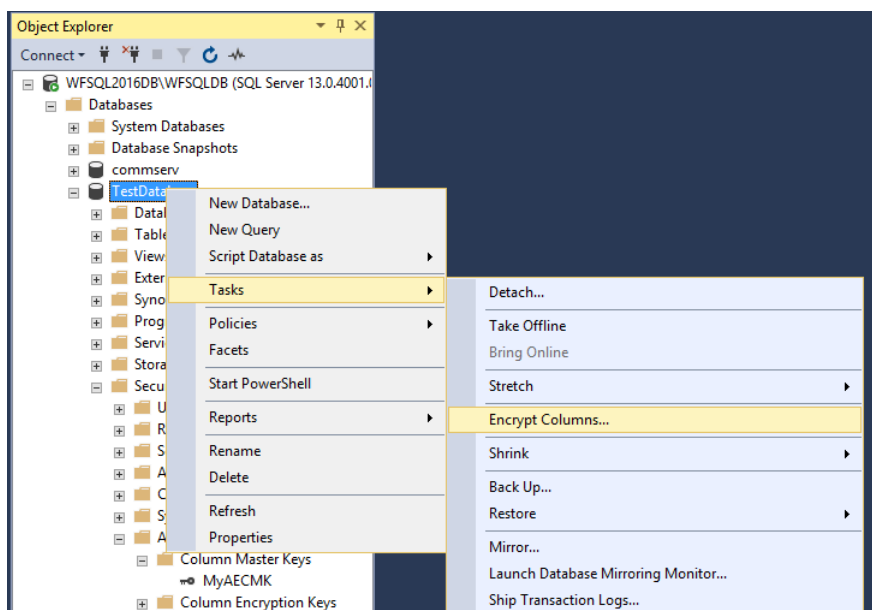


Figure 34: Select Encrypt Columns...

From the Introduction screen, select “Next” to get to the Column Selection page. Click on the field “Encryption Type” to enter your preferred option for this value.

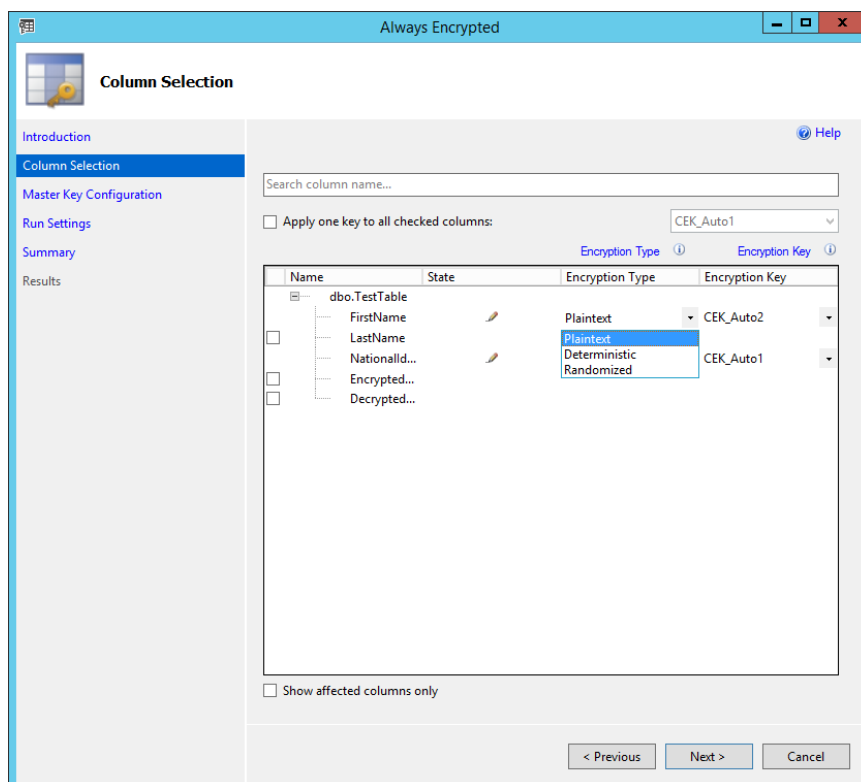


Figure 35: Choose the option - Plaintext

From the drop down list select “Plaintext” then click “Next”.

As there is no key to configure this time click “Next” to proceed straight to the **Run Settings** page. If the database is live at this point, you should first take it off-line before proceeding to remove the column encryption.

Select “Proceed to finish now” and click Next.

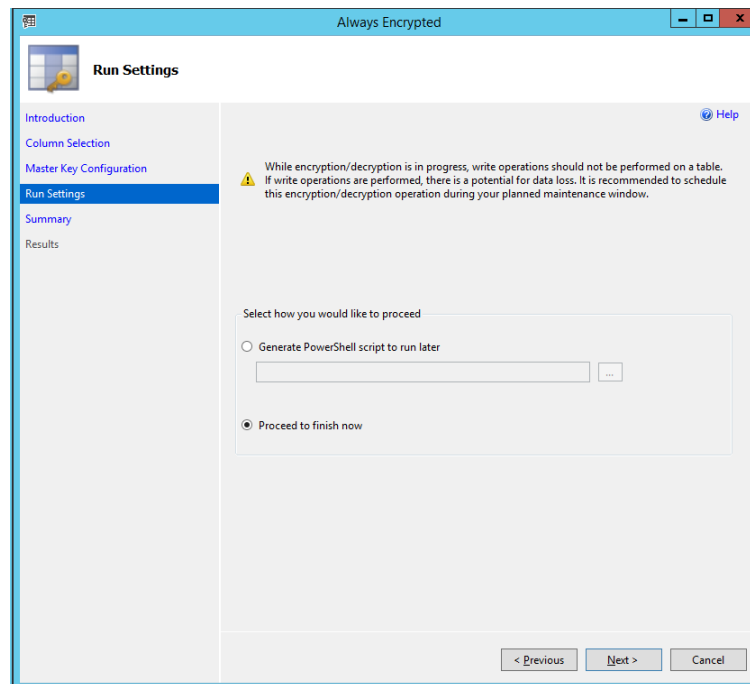


Figure 36: Confirm that database is off-line

The following page will provide a review summary for the requested operations.

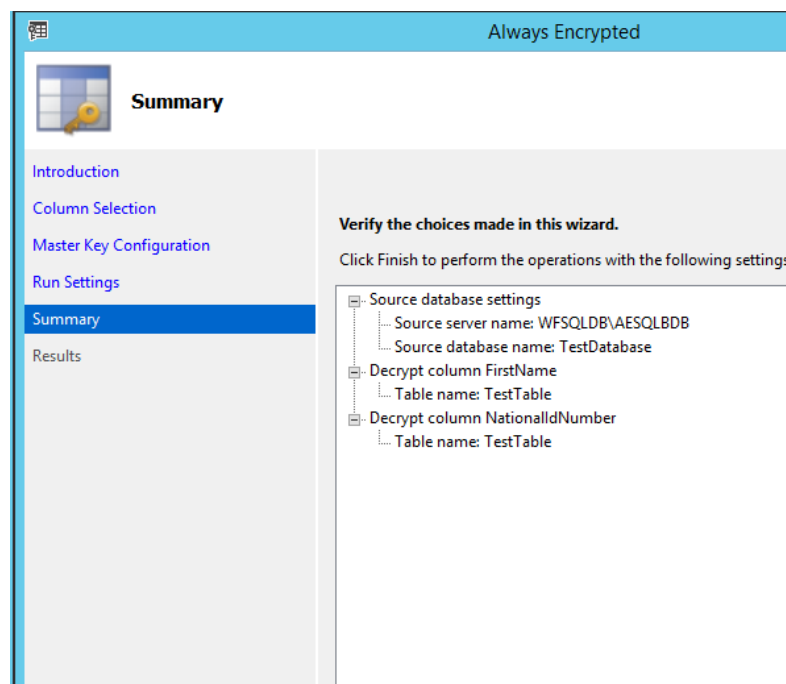


Figure 37: Review column decryption state

Check to ensure that the correct Decrypt column(s) are listed and click “Finish”. The “Performing encryption operations” should show as “Passed”.




Summary	Summary:				
Results	<table> <tr> <th>Task</th><th>Details</th></tr> <tr> <td>  Performing encryption operations </td><td> Passed </td></tr> </table>	Task	Details	 Performing encryption operations	Passed
Task	Details				
 Performing encryption operations	Passed				

Figure 38: Successfully removed Always Encrypted column encryption

You have successfully removed Always Encrypted column encryption from your database. When you next log into the database you can remove the **Column Encryption Setting = enabled** string from the “Additional Connection Parameters” field of the database login screen. When you now view your database table via, “Select Top 1000 Rows” you should see all columns in plaintext (i.e. an unencrypted state).

Always Encrypted using PowerShell: without Role Separation

Install and Configure SqlServer PowerShell module

All sessions must be executed in an Administrator Shell, to facilitate this open Powershell (or PowerShell ISE) by right click and select “Run as Administrator”.

Decide on the level of security required around the running of scripts and change the ExecutionPolicy setting accordingly. Before updating PowerShellGet or PackageManagement, install the latest Nuget provider.

Open a PowerShell session as Administrator and run:

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

Next update PowerShellGet:

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

Then 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.

(Once installed, If you are using PowerShell ISE refresh the Commands pane if you are using PowerShell open a new session. Confirm the install by running:

```
Get-Module -list -Name SqlServer
```

You should see something similar to the output below:

```
Directory: C:\Program Files\windowsPowerShell\Modules

ModuleType Version      Name           ExportedCommands
-----
Manifest    21.0.17152 SqlServer      {Add-SqlColumnEncryptionKeyValue, Complete-SqlColumnMasterKeyRotatio...
```

Figure 39: SqlServer module installed

Install the Thales nCipher CNG provider

Please refer to the section “Install and register the CNG provider” on page 8 of this guide for details on installing and registering the nCipher CNG provider.

Note: if using an existing Security World ensure that the “Use existing Security World” check box is ticked.

The provider location can be found at:

HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Control\Cryptography\Providers\nCipherSecurityWorldKeyStorageProvider

Creating the Always Encrypted Column Master Key using the nCipher KSP

Once you have successfully installed the nCipher CNG Key Storage Provider and registered for use with *Module* protection you can begin to configure Always Encrypted.

Note: Always ensure that you check and confirm all values are adjusted according to your environment; the values in this integration guide are example values only. The bespoke values have been highlighted throughout in **orange**.

Generate a CNG RSA key pair for use as a Column Master Key:

```
$cngProviderName = "nCipher Security world Key Storage Provider"
$cngAlgorithmName = "RSA"
$cngKeySize = 2048 # Recommended key size for Always Encrypted column master keys
$cngKeyName = "AECMK" # 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 will generate a 2048 bit RSA key pair with Name AECMK (highlighted **orange**). The resulting key is encrypted whilst in the HSM and then pushed to the requesting On-Premise Client server where it is stored as an Application Key Token in the %NFAST_KMDATA%\local folder (C:\ProgramData\NCipher\Key Management Data\local).

Next invoke the > New-SqlCngColumnMasterKeySettings cmdlet:

```
## Specify the Column Master Key settings for importing into the database:
$CmkSettings = New-SqlCngColumnMasterKeySettings -CngProviderName "nCipher Security world Key
Storage Provider" -KeyName "AECMK"
```

Then create the Column Master Key via:

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

Creating the Column Encryption Key

Once the Column Master Key has been successfully generated create a Column Encryption Key using the example below:

```
New-SqlColumnEncryptionKey -Name "CEK" -ColumnMasterKeyName "AECMK" -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) this is achieved by calling the EncryptColumnEncryptionKey method of the SqlColumnEncryptionCngProvider provider class.

Encrypting Columns with 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 values highlighted in **Orange** to those suitable for your database name and data columns that you want to encrypt.

–EncryptionType values are one of either:

- Deterministic
- Randomized
- Plaintext (only available to revert encrypted columns to an unencrypted state)

```
#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].[TestTable]
$encryptionChanges += New-SqlColumnEncryptionSettings -ColumnName
dbo.TestTable.NationalIdNumber -EncryptionType Randomized -EncryptionKey "CEK"

Set-SqlColumnEncryption -ColumnEncryptionSettings $encryptionChanges -InputObject $smoDatabase
```

Remove Always Encrypted Column Encryption

To remove column encryption from previously encrypted column data, replace -EncryptionType value (i.e. either Randomized or Deterministic) with the string **Plaintext** and execute.

N.B. If the database is live at this point, you should first take it off-line before proceeding to remove the column encryption.

```
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].[TestTable]
$encryptionChanges += New-SqlColumnEncryptionSettings -ColumnName
dbo.TestTable.NationalIdNumber -EncryptionType Plaintext

Set-SqlColumnEncryption -ColumnEncryptionSettings $encryptionChanges -InputObject $smoDatabase
```

The *Always Encrypted* encrypted data will revert to plaintext. (If your database is protected by TDE then the data is still being encrypted whilst at rest). When you next log into the database you can remove the **Column Encryption Setting = enabled** string from the “Additional Connection Parameters” field of the database login screen. When you now view your database table via, “Select Top 1000 Rows” you should see all columns in plaintext (i.e. an unencrypted state).

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 Key(s) (CMK)

Query the encrypted columns

In order to successfully query the encrypted columns ensure that you connect to the database with the correct connection parameters. You may, upon first time connection, receive the “Parameterization for Always Encrypted” prompt. Click on “Enable” to proceed.

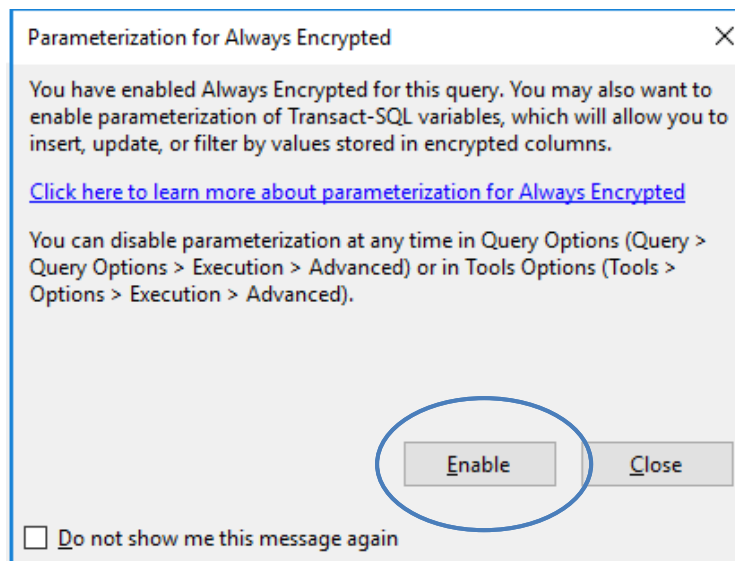


Figure 40: Parameterization for Always Encrypted

When initiating a connection to the database, select “Options” -> Select “Additional Connection Parameters” and enter: *Column Encryption Setting = enabled* in the provided field.

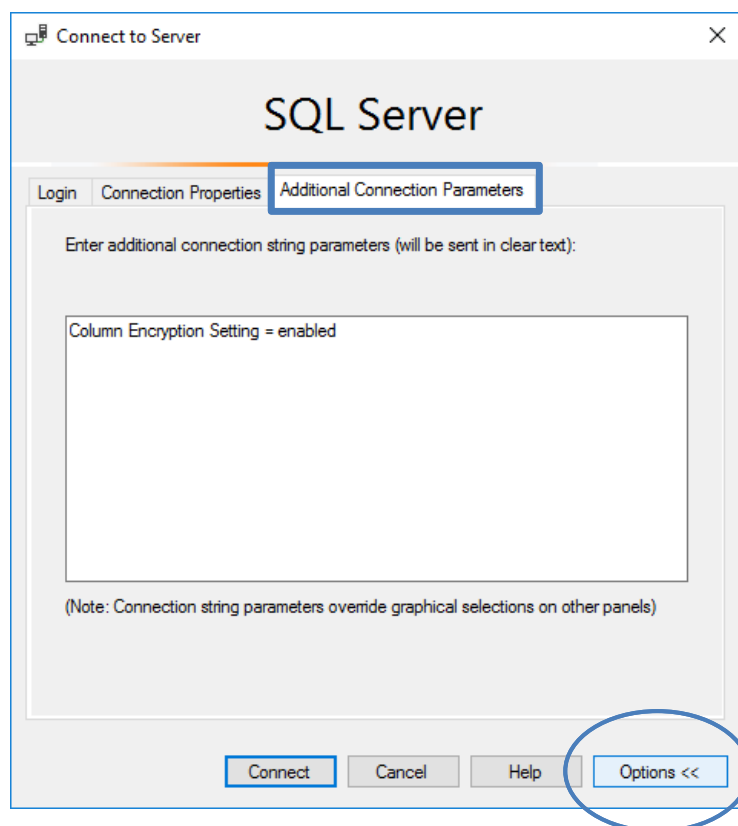


Figure 41: Adjust Connection Parameters

Always Encrypted using PowerShell: with Role Separation

Install and Configure SqlServer PowerShell module and verify KSP

All sessions must be executed in an Administrator Shell, to facilitate this open Powershell (or PowerShell ISE) by right click and select "Run as Administrator". The following operations need to be run on all SQL Always Encrypted servers and the Database server(s) in order to align all with the latest SqlServer package.

Note: Always ensure that you check and confirm all values are adjusted according to your environment; the values in this integration guide are example values only. The bespoke values have been highlighted throughout in **orange**.

Before updating PowerShellGet or PackageManagement, install the latest Nuget provider.

Open a PowerShell session as Administrator and run:

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

Next update PowerShellGet:

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

Then 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.

Once installed, If you are using PowerShell ISE refresh the Commands pane if you are using PowerShell open a new session. Confirm the install by running:

```
Get-Module -list -Name SqlServer
```

You should see the following reported:

```
Directory: C:\Program Files\WindowsPowerShell\Modules
ModuleType Version      Name           ExportedCommands
-----
Manifest    21.0.17152 SqlServer      {Add-SqlColumnEncryptionKeyValue, Complete-SqlColumnMasterKeyRotatio...
```

Verify that the Thales CNG KSP is installed correctly:

```
cnglist.exe -list-providers
Microsoft Key Protection Provider
Microsoft Passport Key Storage Provider
Microsoft Platform Crypto Provider
..
..
nCipher Primitive Provider
nCipher Security world Key Storage Provider
```

Figure 42: Verify Thales KSP is installed

For the purpose of this guide when integrating with role separation, roles are defined as Duty Role. The table below shows the separation and function of these duty roles with reference to Security Administrator and Database Administrator.

Process	Duty Role
Generating the CMK	Security Administrator
Generating / encryption of CEK	Security Administrator
Defining the CMK and CEK in the database	Database Administrator
Encrypt database columns with CEK	Security Administrator

Creating the Always Encrypted Column Master Key using the nCipher KSP

Once you have successfully installed the nCipher CNG Key Storage Provider and registered for use with *Module* protection you can begin to configure Always Encrypted.

The Security Officer must have administrator rights on the Client Server being configured to use Always Encrypted. The following sections are divided between the [Security Administrator](#) and the [Database Administrator](#). The DBA should not have access to the Client server.

Duty Role: Security Administrator

Confirm that the SqlServer module is present by running:

```
Get-Module -list -Name SqlServer
```

```
Directory: C:\Program Files\WindowsPowerShell\Modules
```

ModuleType	Version	Name	ExportedCommands
Manifest	21.0.17178	SqlServer	{Add-SqlColumnEncryptionKeyValue, Complete-SqlColumnMasterKeyRotation, Get-SqlColumnEncryptionKey, Get-SqlCo...

Generate a CNG RSA key pair for use as a Column Master Key: Choose a suitable name for the Column Master Key (\$cngKeyName highlighted **orange** in the example, below) for your particular deployment.

```
$cngProviderName = "nCipher Security World Key Storage Provider"
$cngAlgorithmName = "RSA"
$cngKeySize = 2048 # Recommended key size for Always Encrypted column master keys
$cngKeyName = "AECMK" # 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 will generate a 2048 bit RSA key pair. The resulting key is encrypted whilst in the HSM and then pushed to the requesting On-Premise Client server, where it is stored as an Application Key Token in the %NFAST_KMDATA%\local folder (C:\ProgramData\nCipher\Key Management Data\local).

Confirm that the key was successfully generated and exists in the Thales %KMDATA_LOCAL% directory by running **>nfkminfo -k** this will list the available keys associated with the Security World:

```
nfkminfo.exe -k
Key list - 1 keys
  AppName  caping          Ident  s-1-5-21-1277476411-3880915791-1682396242-1002--
7cbbd9d5477b6d2ed4b6df83e3fa50ac3745b855

PS C:\WINDOWS\system32> nfkminfo.exe -k caping s-1-5-21-1277476411-3880915791-1682396242-1002--
7cbbd9d5477b6d2ed4b6df83e3fa50ac3745b855
Key AppName  caping  Ident  s-1-5-21-1277476411-3880915791-1682396242-1002--
7cbbd9d5477b6d2ed4b6df83e3fa50ac3745b855
name          "AECMK"
hash          9f215d32a765a708e4fcb92678ce43d0f475955b
recovery      Enabled
protection    Module
```

The Security Administrator can now create the `SqlCngColumnMasterKeySettings` object which will reference the CMK application key token created using the nCipher KSP

```
$CmkSettings = New-SqlCngColumnMasterKeySettings -CngProviderName "nCIPHER Security world Key
Storage Provider" -KeyName AECMK
```

Finally invoke the `New-SqlColumnEncryptionKeyEncryptedValue` cmdlet. This will produce the encrypted value of the CMK which will need to be passed to the Database Administrator and is required for generating a Column Encryption Key.

```
New-SqlColumnEncryptionKeyEncryptedValue -TargetColumnMasterKeySettings $CMKSettings

0x01620000016E00630069007000680065007200200073006500630075007200690074007900200077006F0072006C0
0640020006B00650079002000730074006F0072006100670065002000700072006F00760069006400650072002F0061
00650063006D006B00989CD11831C74395E380A8F16A251014D03E98390BD8AF12252062B653AE80A00C7CFA1FC284
914A38CBA659796C55CF59B131BDC1BF8A862B032F14303B2E6EF1BAC2441EEB4B10CBF9DDBB6523E34E64F7EE0B3F6A
538228EF1964C5E2A1E4ED8056700AA2B25BFB9834679C99A84D7B275F0945429E97639558F79EB3EB363497857BBB6
21C2D8FC822464FCF1F2178A24087F2C5E9ACBECF02DB495D4357C8ED886F1CC8A3070BF0FA9F430389D738AA2CEE1
7880EE7210CA18437FA3FC6333075A33F286239CDA869B4C017F3ADA3D67F7ECB6ECE5F58F6815648665DD36A67FC4CA
B4D3D699130D33852B642A6E6ED2B2FE1692010943E3252530727306035822BDC2CADBDAC13F48748686BADAD72086A
505BA57D482C904A3BE39EE603B7CC26398D5CE589AB6597EA5D577C28C34BC6AA96F17B69F1F053BC148B763C27CD
EE1307C9A46F12CFBC8CF309A417364BFA78C5D9A8FEFC877FCDCC4ACC06AF750436A962C757BAC3ADE9CB3C0DAEE78
002ACE2F36DCF9C1BAE69B39053875910CE4F77E51450C844CC32DF062309CC9B2F29F92A8DDAD883B632C8D3F36620
318175B93763DEF696D5EB9DC37F51655E181207165A896AEFA60271B705B2883BD96BEC6487454A65EFE9D7F48AF5
4FFE1B7FEB0598AC47CA306B23201B8B9B89DFAA574688CA18F5B16FFFAFAF9FA87072CE0E853675E2369CE3ED912F6
7
```

Duty Role: Data Base Administrator

Open PowerShell ISE as Administrator and run the following in order to create a SqlCngcolumnMasterKeySettings object that contains information about the location of your column master key. Ensure that the Sql Server module has been imported.

```
New-SqlCngColumnMasterKeySettings -CngProviderName "nCipher Security world key storage  
Provider" -KeyName AECMK  
  
KeyStoreProviderName KeyPath  
-----  
MSSQL_CNG_STORE      nCipher Security world Key Storage Provider/AECMK
```

```
$CmkSettings = New-SqlCngColumnMasterKeySettings -CngProviderName "nCipher Security world Key  
Storage Provider" -KeyName AECMK
```

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

You can confirm the presence of the newly imported CMK using Object Explorer in SSMS.

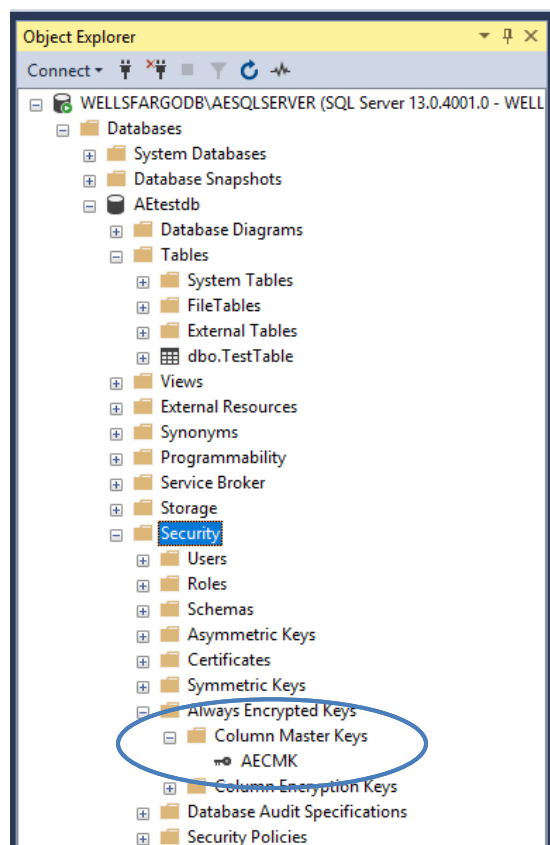


Figure 43: CMK imported into Database

The New-SqlColumnEncryptionKey cmdlet creates a column encryption key object in the form of an “Encrypted Value” in the database. A Column Encryption Key object encapsulates an encrypted value of a symmetric cryptographic key, i.e. the Column Encryption Key; this can be subsequently used to encrypt database columns using the Always Encrypted feature. When attempting to copy the encrypted value, ensure that the entire value is written to a single line, appended with the path to the database.

```
New-SqlColumnEncryptionKey -Name "AECEK1" -ColumnMasterKeyName "AECMK" -EncryptedValue
0x01620000016E00630069007000680065007200200073006500630075007200690074007900200077006F0072006C0
0640020006B00650079002000730074006F0072006100670065002000700072006F00760069006400650072002F0061
00650063006D006B004BD6CACB568A8DFB91AF92CB6AD25C2AEF93895BE3E658D2E7280BB799BD0DBB3B3F38B0E3209
55CC8D9705D42CC8E2D022F7A5839DE6ACF133EFACE9DA7CDF49AB14F21491B06F9156E9509735328567C603A71A483
8B4B8E746A413A2391EDCDEC7F6DFBE9E3DECA579797DFB025F5771330892CCD6AA60FD89F898340E731EBAC43FDA2
40E0FEA6471B2B772BE9238DCEBF96A2F421F7AB23F4EEDFC28CA5D91244BCA5D4ED9B9E53F657E4F32A4D0CA8D62DF
67FEC648224AB3B0A9DDF223B8C82CCEC64E68FA27ED63D4212552BE5C3E69C79F4485C743165C3375909BAE4A11FF1
FBCF69C2D791E634B86636C921F71400E02565B1E0C1E63C01F6C2699CEE2EB6DF3F29BB0B253B65FD0E6741BA0FA71
50330D3BC16357895687433DFBD20027BC133841E02072E2409F543ABB5386B3BA77D4959DA2631B80451868186BCDD
EB781ECDAC567C62DAC8169F7A0951D99A943AC06E8CD88DD9174C27B9FD2A49E37C3555DAC542DEC19582F10628A9B
49FBDE3520BAA684580CF8E51C99987ED47498E803E40793411737E3969B40A676FF5513054569ADB3B0B97B1715FD
3683D5E537E51413A3563DBEA9EDA6F895565EF1793FF3642206F362582153F382AA107C9BA4A3AC619AAB4A3CF6AB5
D8D4DCC1389DF54CBF60B8F6882152DD95215DE8371F3632E1406CC9BD0449CDA2212DEA8D173CFFD8CC04234151

Name
----
AECEK1
```

The screenshot above shows the output confirming the creation of a new CEK, called “AECEK1” this was generated and then encrypted using the encapsulated CMK metadata provided to the DBA by the Security Administrator. (It is advisable to use a word editor to align the encrypted value to a single line before copying to the PowerShell CLI)

Note: When adding the –Path you should append this after the encrypted value:

```
.....C04234151 -Path SQLSERVER:\SQL\server_name\DEFAULT\Databases\your_database
```

You can confirm the presence of the CEK using SSMS Object Explorer to view the new Always Encrypted key.

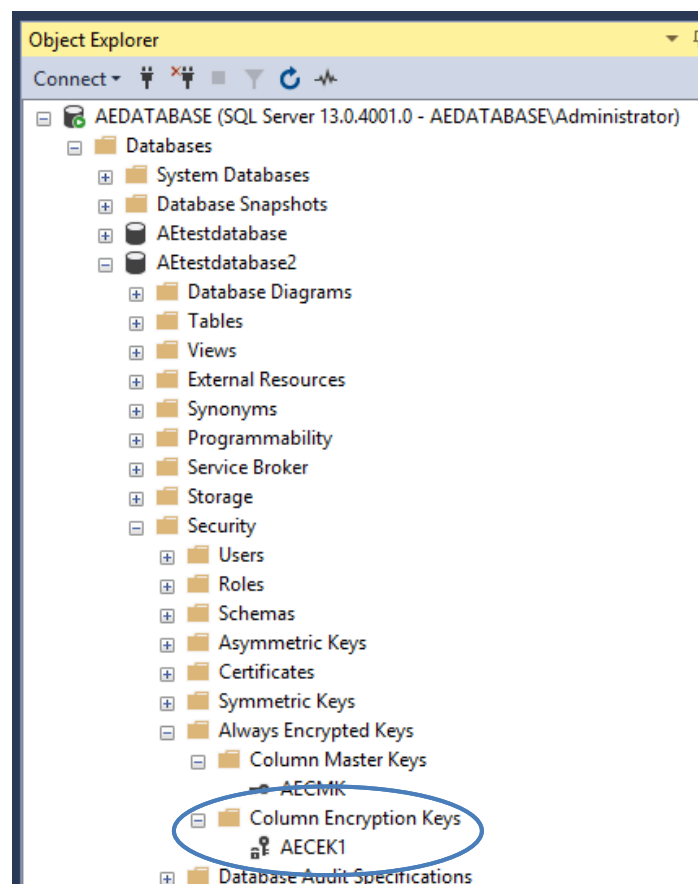


Figure 44: CEK creation

This concludes role separation activities performed by the Data Base Administrator. The Security Administrator can now connect to the Database and use the defined keys to encrypt the desired columns using Always Encrypted.

Duty Role: Security Administrator

The Security Administrator can now connect to the database either through the SQL Server Management Studio or PowerShell and use the provisioned key Metadata to encrypt the required columns.

Encrypt Columns using SSMS

To encrypt a column using SSMS please refer to the chapter on “Always Encrypted using SSMS” this can be found on page 14 of this document.

To encrypt columns using PowerShell please refer to the section below.

Encrypt Columns using PowerShell (ISE)

Note: In order to encrypt a column using a CEK the user of the database must have the following “effective” permissions:

ALTER ANY COLUMN MASTER KEY

ALTER ANY COLUMN ENCRYPTION KEY

VIEW ANY COLUMN MASTER KEY DEFINITION

VIEW ANY COLUMN ENCRYPTION KEY DEFINITION

These permissions can be found by right clicking on the relevant database and selecting “Properties” then select “permissions” (The example below has been edited for convenience of illustration).

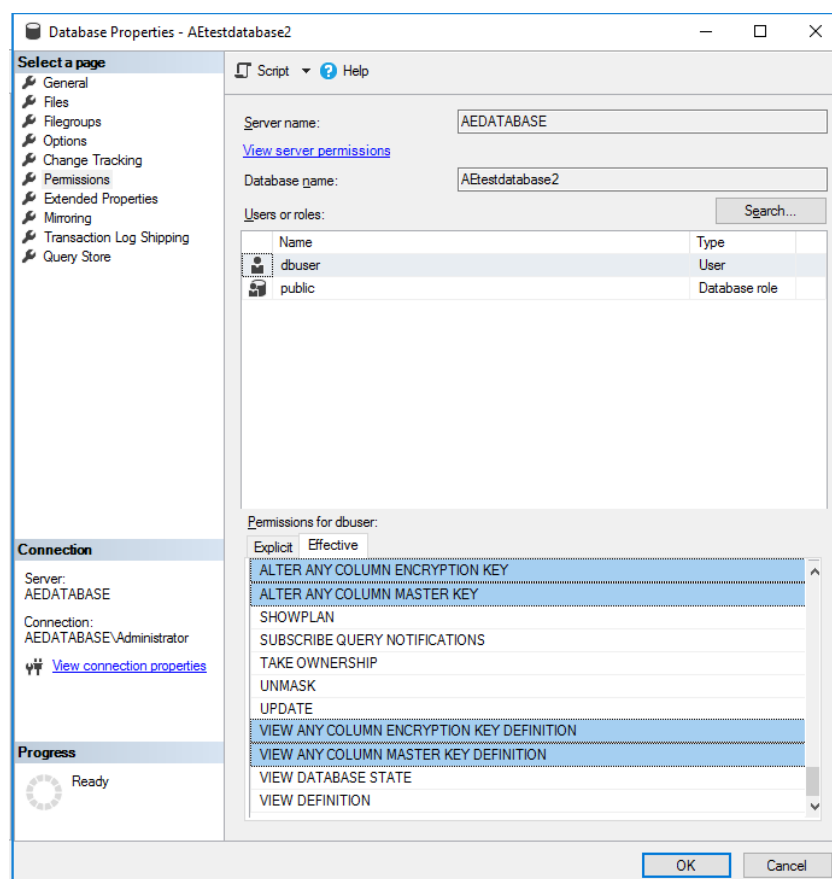


Figure 45: Permissions Status

To encrypt a column using the pre-defined Column Encryption Key (given in the example as AECEK1) use one of the following for `-EncryptionType` value.

Available Options are:

- Deterministic
- Randomized
- Plaintext (only available to revert encrypted columns to an unencrypted state)

Run the below with the relevant server name and database inserted appropriately and your chosen table / column intended for encryption, specify also the Column Encryption Key to use.

```
# 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].[TestTable]
$encryptionChanges += New-SqlColumnEncryptionSettings -ColumnName
dbo.TestTable.NationalIdNumber -EncryptionType Deterministic -EncryptionKey "AECEK1"

Set-SqlColumnEncryption -ColumnEncryptionSettings $encryptionChanges -InputObject $smoDatabase
```

Note: As the Column Master Key Application Key Token only exists in the NFAST-KMDATA%\local directory of a participating Always Encrypted client server, the facility to decrypt a column is dependent on the presence of both the CMK Application Key Token in the NFAST-KMDATA%\local directory and the availability of a correctly configured HSM, thereby rendering all encrypted data available only to those computers configured with and with access too, these requisite resources.

Remove Always Encrypted Column Encryption

To remove column encryption from previously encrypted column data, replace -EncryptionType value (i.e. either Randomized or Deterministic) with the value **Plaintext** and execute.

N.B. If the database is live at this point, you should first take it off-line before proceeding to remove the 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=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].[TestTable]
$encryptionChanges += New-SqlColumnEncryptionSettings -ColumnName
dbo.TestTable.NationalIdNumber -EncryptionType Plaintext

Set-SqlColumnEncryption -ColumnEncryptionSettings $encryptionChanges -InputObject $smoDatabase
```

The *Always Encrypted* encrypted data will revert to plaintext. (If your database is protected by TDE then the data is still being encrypted whilst at rest). When you next log into the database you can remove the **Column Encryption Setting = enabled** value from the "Additional Connection Parameters" field of the database login screen. When you now view your database table via, "Select Top 1000 Rows" you should see all columns in plaintext (i.e. an unencrypted state).

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 Key(s) (CMK)

Query the encrypted columns

In order to successfully query the encrypted columns ensure that you connect to the database with the correct connection parameters. You may, upon first time connection, receive the “Parameterization for Always Encrypted” prompt. Click on Enable to proceed.

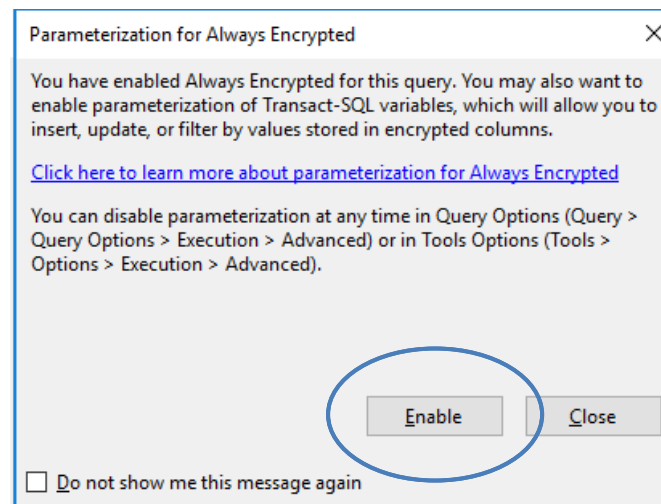


Figure 46: Parameterization for Always Encrypted

When initiating a connection to the database, select “Options” -> Select “Additional Connection Parameters” and enter: *Column Encryption Setting = enabled* in the provided field.

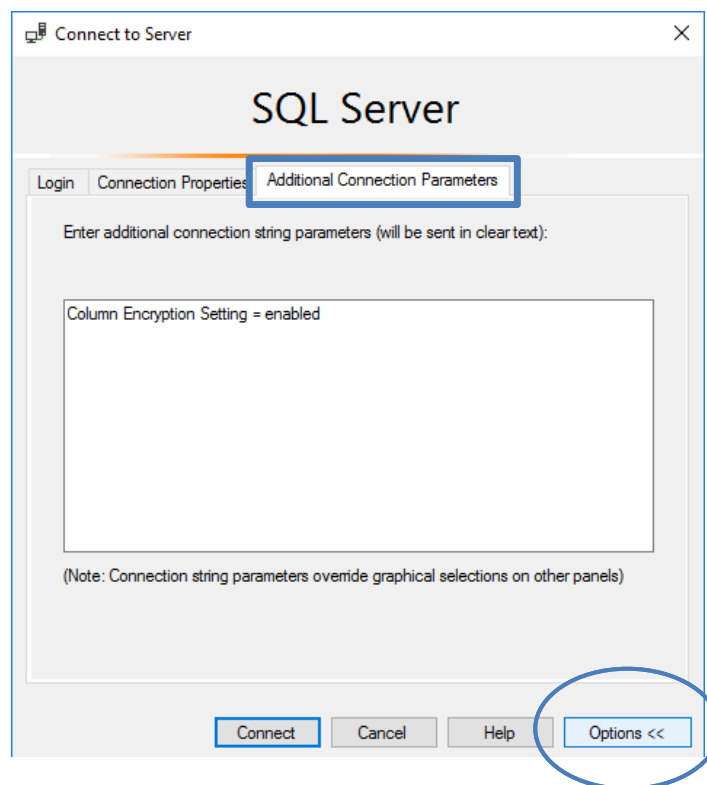


Figure 47: Adjust Connection Parameters

Glossary of PowerShell SqlServer CMDlets

The following list shows the currently available Always Encrypted PowerShell cmdlets:

PowerShell CMDlet	Description
Add-SqlColumnEncryptionKeyValue	Adds a new encrypted value for an existing column encryption key object in the database.
Complete-SqlColumnMasterKeyRotation	Completes the rotation of a column master key.
Get-SqlColumnEncryptionKey	Returns all column encryption key objects defined in the database, or returns one column encryption key object with the specified name.
Get-SqlColumnMasterKey	Returns the column master key objects defined in the database, or returns one column master key object with the specified name.
Invoke-SqlColumnMasterKeyRotation	Initiates the rotation of a column master key.
New-SqlAzureKeyVaultColumnMasterKeySettings	Creates a SqlColumnMasterKeySettings object describing an asymmetric key stored in Azure Key Vault.
New-SqlCngColumnMasterKeySettings	Creates a SqlColumnMasterKeySettings object describing an asymmetric key stored in a key store supporting the Cryptography Next Generation (CNG) API.
New-SqlColumnEncryptionKey	Creates a new column encryption key object in the database.
New-SqlColumnEncryptionKeyEncryptedValue	Produces an encrypted value of a column encryption key.
New-SqlColumnEncryptionSettings	Creates a new SqlColumnEncryptionSettings object that encapsulates information about a single column's encryption, including CEK and encryption type.
New-SqlColumnMasterKey	Creates a new column master key object in the database.
New-SqlCspColumnMasterKeySettings	Creates a SqlColumnMasterKeySettings object describing an asymmetric key stored in a key store with a Cryptography Service Provider (CSP) supporting Cryptography API (CAPI).
Remove-SqlColumnEncryptionKey	Removes the column encryption key object from the database.
Remove-SqlColumnEncryptionKeyValue	Removes an encrypted value from an existing column encryption key object in the database.
Remove-SqlColumnMasterKey	Removes the column master key object from the database.
Set-SqlColumnEncryption	Encrypts, decrypts or re-encrypts specified columns in the database.

Alternatively, the full list, along with any additions to the SqlServer module released after this guides publication, can be found via the link below:

<https://blogs.technet.microsoft.com/dataplatforminsider/2016/06/30/sql-powershell-july-2016-update/>

Feature Details

The information below was taken from the Microsoft website on relational databases;

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

For further information on Always Encrypted operational capabilities please consult the website via the above URL.

- 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

Troubleshooting

Common returned errors when encrypting columns:

Error	Resolution
<pre>Set-SqlColumnEncryption : Object reference not set to an instance of an object. At line:16 char:1 + Set-SqlColumnEncryption -ColumnEncryptionSettings \$encryptionChanges ... + ~~~~~ + CategoryInfo : InvalidOperation: (:) [Set-SqlColumnEncryption], NullReferenceException + FullyQualifiedErrorId : EncryptionError,Microsoft.SqlServer.Management.PowerS hell.AlwaysEncrypted.SetColumnEncryption</pre>	<p>Check the correct CEK is referenced in the call and re-submit.</p>
<pre>Get-SqlDatabase : Failed to connect to server <server_name>. At line:7 char:16 + \$smoDatabase = Get-SqlDatabase -ConnectionString \$SqlConnectionString + ~~~~~ + CategoryInfo : NotSpecified: (:) [Get-SqlDatabase], ConnectionFailureException + FullyQualifiedErrorId : Microsoft.SqlServer.Management.Common.ConnectionFai lureException,Microsoft.SqlServer.Management.PowerS hell.IaaS.GetSqlDatabaseCommand</pre>	<p>Check the Database server name is correct and available to the network.</p>
<pre>Set-sqlColumnEncryption : Cannot validate argument on parameter 'InputObject'. The argument is null or empty. Provide an argument that is not null or empty, and then try the command again. At line:17 char:83 + ... ColumnEncryptionSettings \$encryptionChanges - InputObject \$smoDatabase + ~~~~~ + CategoryInfo : InvalidData: (:) [Set-SqlColumnEncryption], ParameterBindingValidationException + FullyQualifiedErrorId : ParameterArgumentValidationError,Microsoft.SqlServe r.Management.PowerShell.AlwaysEncrypted.SetColumnEn cr yption</pre>	<p>Check database name is correct.</p>

END OF DOCUMENT

About Thales e-Security

Thales e-Security is the leader in advanced data security solutions and services that deliver trust wherever information is created, shared or stored. We ensure that the data belonging to companies and government entities is both secure and trusted in any environment – on-premise, in the cloud, in data centers or big data environments – without sacrificing business agility. Security doesn't just reduce risk, it's an enabler of the digital initiatives that now permeate our daily lives – digital money, e-identities, healthcare, connected cars and with the internet of things (IoT) even household devices. Thales provides everything an organization needs to protect and manage its data, identities and intellectual property and meet regulatory compliance – through encryption, advanced key management, tokenization, privileged user control and high assurance solutions. Security professionals around the globe rely on Thales to confidently accelerate their organization's digital transformation. Thales e-Security is part of Thales Group.

Follow us on:

