nShield Database Security Option Pack

nDSOP v2.1.0 User Guide

10 April 2024

Table of Contents

1. Introduction

This guide applies to the nShield Database Security Option Pack, which provides data-at-rest encryption for sensitive information held by Microsoft SQL Server.

The product works in combination with Entrust nShield Hardware Security Modules (nShield HSMs), nShield Security World Software, and Enterprise Editions of Microsoft® SQL Server®, to provide a high quality SQL Extensible Key Management (SQLEKM) provider. It is designed to be integrated into a Microsoft SQL Server database infrastructure with minimal disruption.

The nShield SQLEKM provider supports Transparent Data Encryption (TDE) and Cell-Level Encryption (CLE), and the concurrent use of both TDE and CLE.

1.1. Product configurations

For details of supported and tested versions, see the Release Notes available at https://nshieldsupport.entrust.com/hc/en-us/sections/360001115837-Release-Notes.

1.2. Supported nShield functionality

You can access the following functionality when you integrate an nShield HSM with Microsoft SQL Server:

Functionality	Support
Key Generation	Yes
1 of N Card Set	Yes
K of N Card Set	No
Softcards	Yes
Module Only Key	No
Key Management	Yes
Key Recovery	Yes
Key Import	Partial (see note 1)
Load Balancing	Yes
Fail Over	Yes
FIPS 140-2 Level 3 Security Worlds	Yes

Functionality	Support
Common Criteria (CC) CMTS Security Worlds	Yes

[1] Please see Importing keys.

1.3. Contacting Support

To obtain support for your product, visit https://nshieldsupport.entrust.com.

2. Overview

This chapter provides an overview of how the Extensible Key Management (EKM) API, as provided for Microsoft SQL Server, can be used to protect databases through encryption. It explains how the nShield Database Security Option Pack supports this by including the security benefits of an nShield HSM and associated nShield Security World software. A brief description of how to perform encryption operations on Microsoft SQL Server using the SQLEKM provider is also given.

Encryption should be part of a wider scheme of security practices to protect your database assets that should take into account any regulatory or legal requirements for data protection. Administration and management of encryption within any organization is a serious issue that requires appropriate training and resources.

Data in transit between a database server and client may not be encrypted. Communication between servers and clients should be independently encrypted to ensure security during data transmission. The encryption schemes described here are designed only to protect data at rest.

Figure 2.1 provides a graphical overview of the cryptographic architecture outlined here.

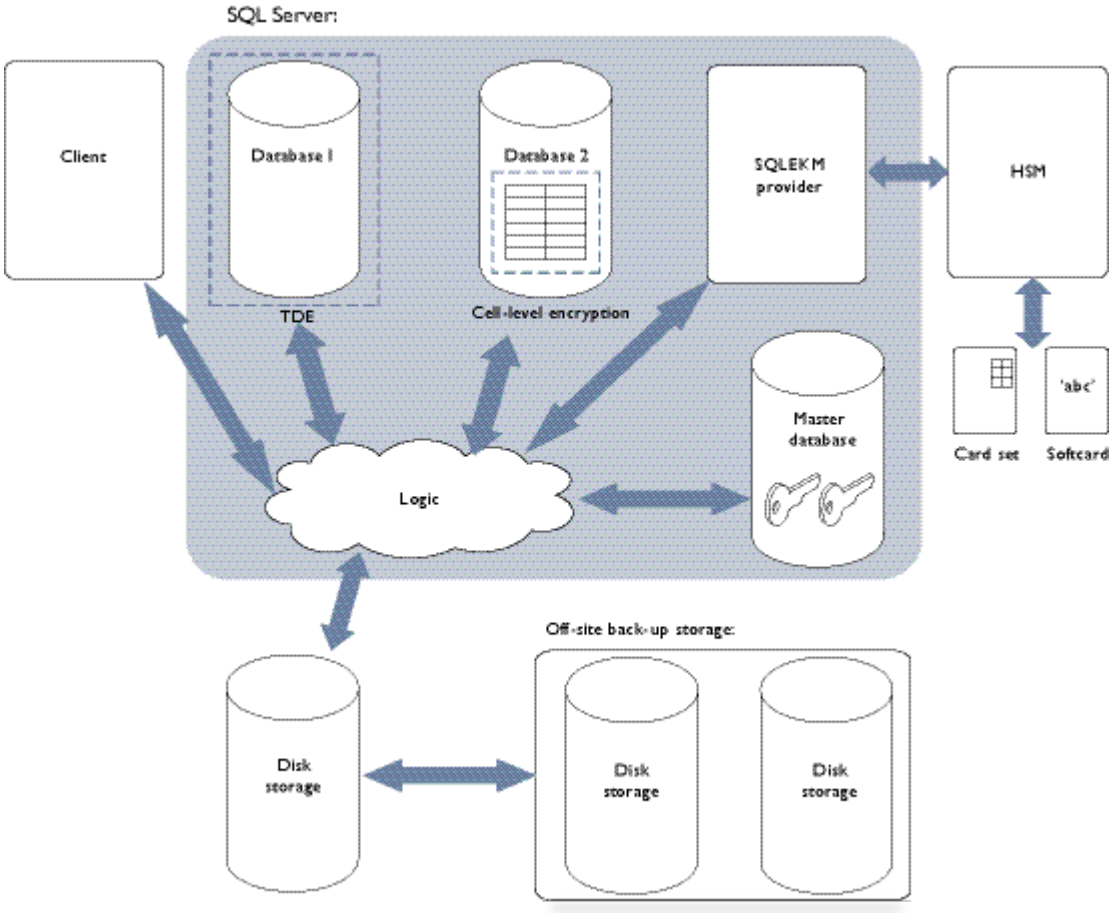

Figure 2.1: Cryptographic architecture

A Microsoft SQL Server service permits the creation of one or more databases. When a client request is made to the SQL Server, it determines which of the databases are the subject of the query, and loads data that is the subject of the query into available memory from disk storage.

From a security perspective, the Microsoft SQL Server supports the use of cryptographic keys to protect its databases. These encryption keys can be used to perform two levels of encryption.

- **Transparent Data Encryption (TDE)** is used to encrypt an entire database in a way that does not require changes to existing queries and applications. A database encrypted with TDE is automatically decrypted when SQL Server loads it into memory from disk storage, which means that a client can query the database within the server environment without having to perform any decryption operations. The database is encrypted again when saved to disk storage. When using TDE, data is not protected by encryption whilst in memory. Only one encryption key at a time per database can be used for TDE.
- To use **Cell-Level Encryption (CLE)**, you must specify the data to be encrypted and the key(s) with which to encrypt it. CLE uses one or more keys to encrypt individual

cells or columns. It gives the ability to apply fine-grained access policies to the most sensitive data in a database. Only the specified data is encrypted: other data remains unencrypted. This mode of encryption can minimize data exposure within the database server and client applications. You can apply CLE to database tables that are also encrypted using TDE. Note that when using CLE, data is only decrypted in memory when required for use. Separate data can be encrypted using different encryption keys within the same data table.

There may be performance trade-offs between speed and security, regarding use of TDE or CLE, but these issues are beyond the scope of this document.

Cryptographic keys can be stored by the database itself, or off-loaded to a SQLEKM provider. Use of a SQLEKM provider is more secure because encryption keys are stored separately from the associated encrypted data. Typically, a SQLEKM provider will also support encryption acceleration and enhanced facilities dedicated to the generation, back up, management and secure protection of the encryption keys. These facilities become more important as the amount of encrypted data, and the number of encryption keys, increases.

Use of the nShield SQLEKM provider in conjunction with an nShield HSM provides the following benefits:

- Ability to store keys from across an enterprise in one place for easy management
- Key retention (rotate keys while keeping the old ones)
- Reduced costs of regulatory compliance
- FIPS certification
- Common criteria certification.

When the nShield HSM(s), Security World and nShield SQLEKM provider software have been correctly set up, the appropriate encryption keys can be made available to a Microsoft SQL Server database. Authorized access to the secure environment of a HSM and encryption keys under its protection is controlled by an Operator Card Set (OCS) or a softcard. To use an OCS or softcard, you must first set up a database credential.

To read from or write to an encrypted database, a user must have *all* of the following:

- An authorized database login, with password, that maps to an appropriate database credential.
- The correct OCS cards, or knowledge of the correct softcard(s).
- The passphrase(s) associated with the OCS cards or softcard(s).
- The nShield Security World holding the encryption keys.

- For CLE, knowledge of the encryption keys in use, and their passwords (if any).
- An nShield HSM with the software to drive it and, if necessary, the authorized administrative mechanisms to load it with the Security World data.
- Knowledge of the appropriate encrypted database to read or write to.

If Security World data (or encryption keys) are lost, they can be securely recovered from a backup as authorized through secure administrative means. It is important to maintain an up-to-date backup of your data.

When use of encryption keys is legitimately made available to the database, the continuing security of data protected by those keys becomes dependent on access offered through SQL Server in accordance with your organization's security policies.

For more information about:

- Configuring the SQLEKM provider to perform encryption operations on SQL Server, see Using the SQLEKM provider.
- Restoration of Security World data from backup, see Disaster recovery.

2.1. Querying encrypted data

When the client sends a query to SQL Server, the SQLEKM provider checks the level of encryption on the database that is the subject of the query. If SQL Server uses a database that employs TDE, the process of loading the assigned encryption keys and encrypting the database when it is stored is done automatically. The reverse decryption operation is also automatic when a TDE encrypted database needs to be used and is loaded into memory. If a database is encrypted using TDE only, this is transparent to the client or user who does not need to be aware of the encryption status or specify any encryption or decryption operations when querying the database. Backup and transaction logs are similarly encrypted.

CLE can be used with or without TDE. In either case, when using CLE the target data must be explicitly encrypted in memory before being stored, or explicitly decrypted after being loaded into memory from storage. You must specify:

- The fields to be encrypted or decrypted.
- The (correct) cryptographic key to be used.

CLE is not automatic. If you use it, you must be aware of the encryption or decryption process.

Note that if TDE is used in combination with CLE, then after the CLE has been performed, the encrypted cells will be additionally encrypted by the TDE process when the data is stored. When the TDE process decrypts, the cells are returned to memory in their original encrypted form and must be decrypted a second time using the appropriate cell-level cryptographic key. The database-level TDE processes remain automatic.

2.1.1. Example queries

The following example queries use a database table of customer information that includes first names, second names and payment card numbers. The queries concern the details of customers whose first names are Joe.

2.1.1.1. Example 1: TDE encryption/decryption only

In this example, the entire database is encrypted with TDE.

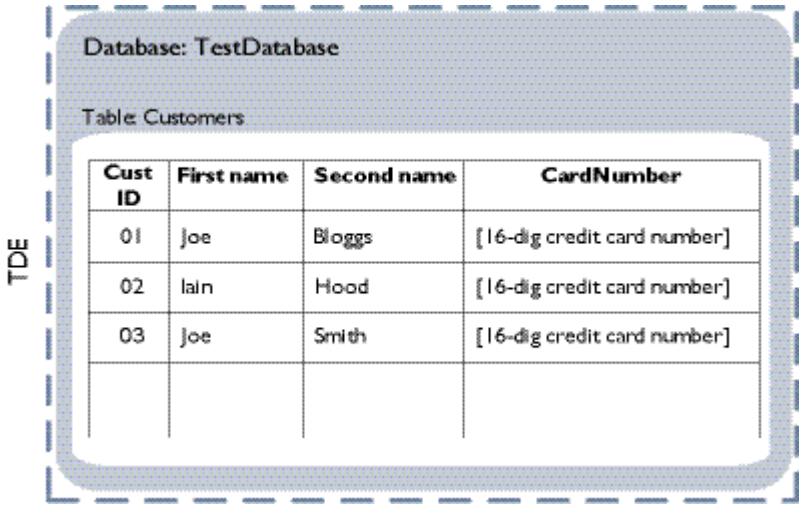

Figure 2.2: TDE encryption/decryption only

The database is decrypted when it is loaded into memory from disk storage. As this happens before the query is performed, the query does not have to specify any decryption operation:

```
USE TestDatabase
SELECT * FROM Customers WHERE
FirstName LIKE ('%Joe%');
```

2.1.1.2. Example 2: TDE combined with CLE/decryption

In this example, the database is encrypted with TDE, and the column of credit card numbers

in the table of customers is additionally protected with CLE.

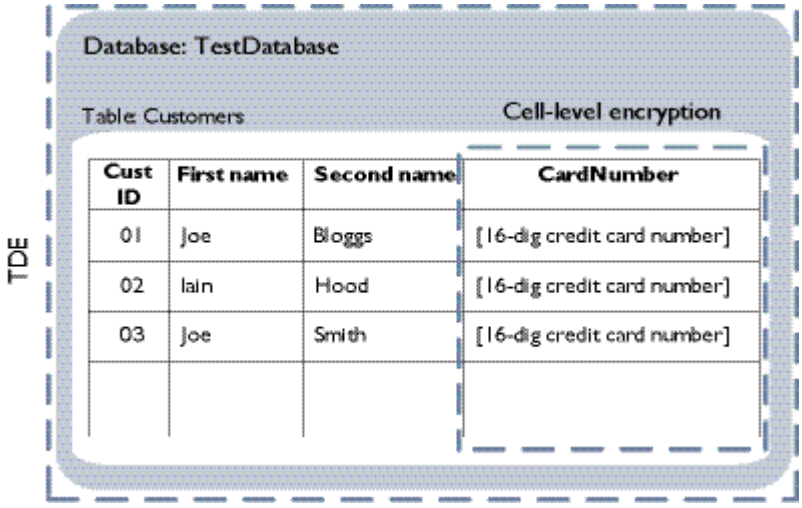

Figure 2.3: TDE and CLE/decryption

The query does not have to take account of TDE on the database because it is automatically decrypted on loading into memory from disk storage before the query is performed. However, the query must specify the (cell-level) decryption of the column of credit card numbers before the details of customers called 'Joe' can be returned.

```
USE TestDatabase
SELECT [FirstName], [SecondName], CAST(DecryptByKey(CardNumber) AS VARCHAR)
AS 'Decrypted card number'
FROM Customers WHERE [FirstName] LIKE ('%Joe%');
```

3. Installation

This chapter describes how to install and setup the nShield Database Security Option Pack for both standalone and clustered deployments.

The installation described here assumes the Microsoft SQL Server software and Entrust nShield Security World software are already installed.

Ensure that all the latest service packs, updates and hotfixes for Microsoft SQL Server software have been added.

A SQL Server login and appropriate permissions are required for all users who wish to install, configure or use the SQLEKM provider. Suitable permissions can be granted by a system administrator according to your company access policy.

To install the SQLKM provider as a stand-alone service, refer to Setting up as stand alone service. To install the SQLKM provider within a database cluster environment, refer to Usage with database failover clusters.

3.1. Setting up as stand alone service

To install the nShield Database Security Option Pack:

1. Log in as Administrator or as a user with local administrator rights.
2. Using the provided installation media, launch `setup.msi` manually.
3. Follow the onscreen instructions. Accept the license terms, and click **Next** to continue.
4. The SQLEKM provider will be installed to `%NFAST_HOME%`. Click **Install** to initiate installation.
5. Click **Finish** to complete the installation.

Before using the SQLEKM provider (see Using the SQLEKM provider), you should confirm that a Security World exists and that the HSM is usable: run `nfkminfo` and confirm that the World state returned shows `Usable` and that the desired `Module n` state (where `n` is the number of the module) shows `0x2 Usable`. For more information, see *"Creating and managing a Security World"* in the *User Guide* for your HSM. Then, create an OCS or softcard (see Security Worlds, key protection and failover recovery).

3.2. Usage with database failover clusters

The nShield SQLEKM provider can function as part of a Microsoft SQL Server database failover cluster. Three typical configurations are shown as examples.

- Two incorporate a two-node failover cluster using a **shared disk**. The active server is the cluster server currently in ownership of the shared drive. See Failover cluster using nShield Solo+ HSMs and Failover cluster using nShield Connect XC HSMs.
- The third example is an AlwaysOn availability group with **no shared disk** for TDE encryption. The active server is the one acting as the primary replica. See Failover cluster with HSMs in an AlwaysOn availability group.

User access to the failover cluster will typically be through a virtual server that will have its own name and IP address.

When deploying in a clustered environment, the same Security World should be used across all HSMs.

If using Java cards, the cardlist file should be the same on all servers.

3.2.1. Failover cluster using nShield Solo+ HSMs

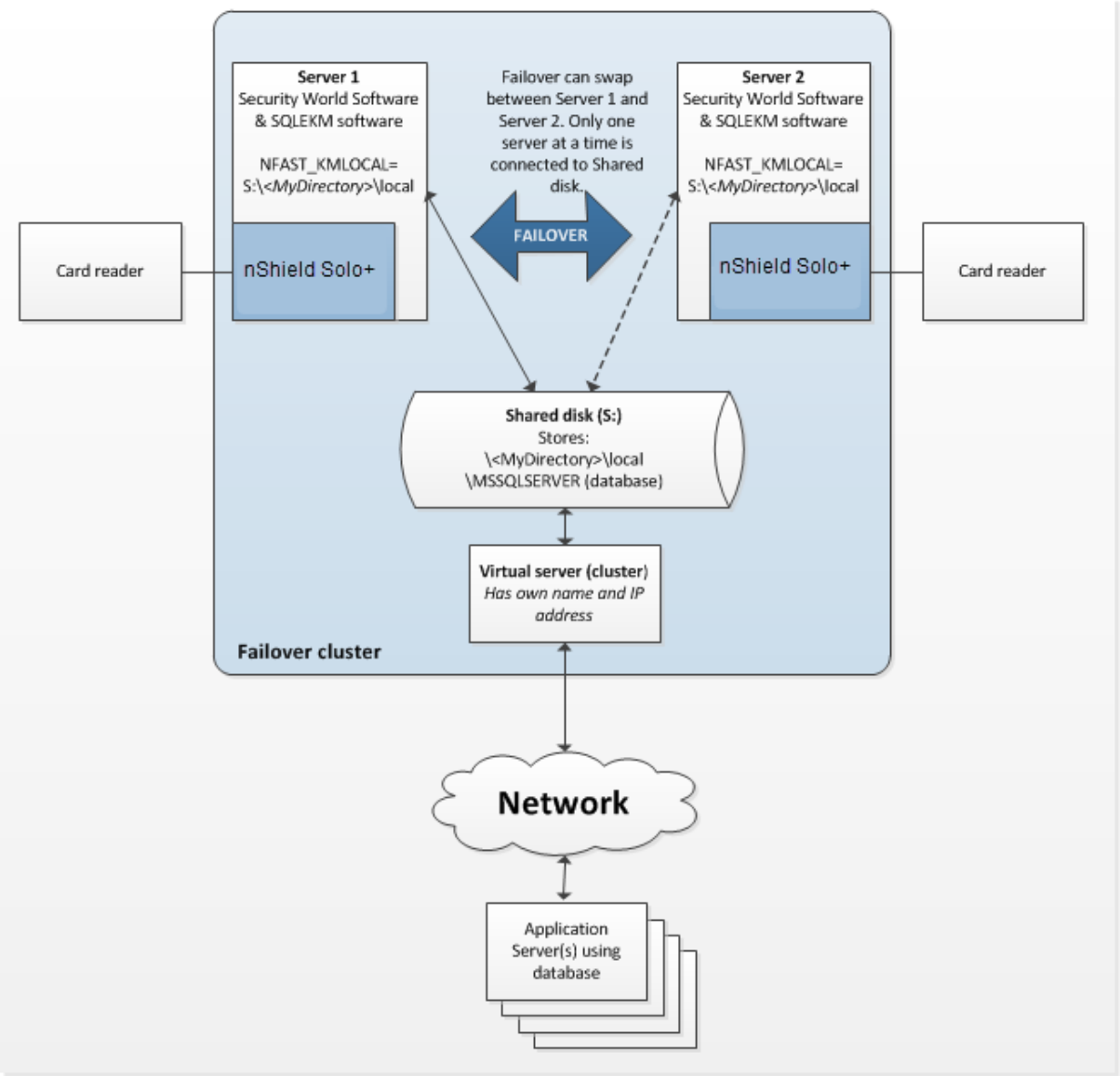

Figure 3.1: SQL Server database failover cluster using nShield Solo+

Figure 3.1 shows a two node database failover cluster example, using nShield Solo+ HSMs. To implement this configuration:

1. On Server 1, complete the installation instructions in Setting up as stand alone service, ensuring a Security World exists.
2. On Server 2, complete the installation instructions in Setting up as stand alone service, but do *not* create a Security World.
3. For the database cluster to function correctly in failover mode, the Security World data must be held in the shared network drive for the cluster. If the shared network drive is `S:` then create the following directory path on that drive, through the active server:

```
S:\<MyDirectory>\local
```

4. On Server 1 and Server 2, do the following:
    a. Create the environment variable `%NFAST_KMLOCAL%` and set its value to that of the shared directory path, e.g. `NFAST_KMLOCAL=S:\<MyDirectory>\local`.

The Security World should already exist on Server 1, and be loaded onto its HSM.

    b. Make Server 1 active in the cluster. From Server 1 the contents of the directory `%NFAST_KMDATA%\local` must be copied to the shared directory `S:\<MyDirectory>\local`.
5. Make Server 2 active in the cluster. Load the Security World onto the HSM. See the *User Guide* for your HSM if you require help.

Before using the SQLEKM provider (see Using the SQLEKM provider), you should confirm that a Security World exists and that the HSM is usable: run `nfkminfo` and confirm that the World state returned shows `Usable` and that the desired `Module n` state (where `n` is the number of the module) shows `0x2 Usable`. For more information, see "Creating and managing a Security World" in the *User Guide* for your HSM. Then, create an OCS or softcard (see Security Worlds, key protection and failover recovery).

3.2.2. Failover cluster using nShield Connect XC HSMs

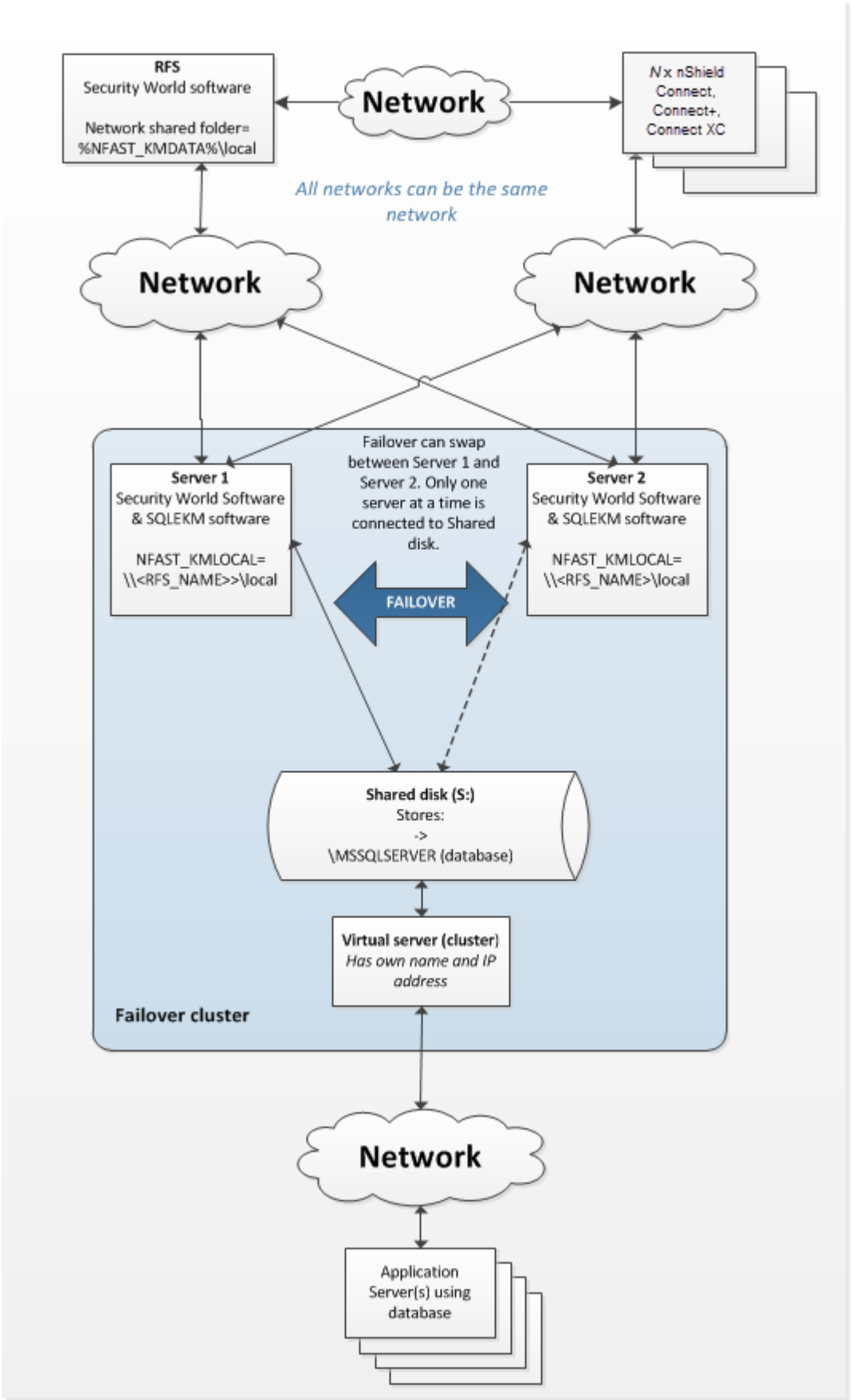

Figure 3.2: SQL Server database failover cluster using nShield Connect HSMs

Figure 3.2 shows a two node database failover cluster example using a shared disk that is configured to use nShield Connect HSMs. You will need a separate host to act as the RFS in this configuration. An example of configuring an AlwaysOn availability group with no shared disk for TDE encryption is given in Failover cluster with HSMs in an AlwaysOn availability group.

In this example, if there is failure of the entire system (for instance a temporary power loss) then the RFS and nShield Connect HSMs should be re-powered before the failover cluster.

To implement this configuration:

1. Install Security World software on the RFS. See the appropriate *User Guide* for your HSM.
2. On the RFS, make the directory `%NFAST_KMDATA%\local` a shared directory that is visible on the network. Grant permissions on the shared network folder for all users of the SQL Server database who will also need to use the SQLEKM provider.

   

   As well as permissions to use the shared folder, the users will also require remote access permissions to the RFS. If your SQL Server process is running as an autonomous service user, this must be granted similar permissions. Check your company security policies before making changes to permissions.

3. On Server 1 and Server 2, complete the installation instructions in Setting up as stand alone service, but do *not* create a Security World.
4. On Server 1 and Server 2, set the system environment variable `%NFAST_KMLOCAL%` to point to the shared network folder on the RFS. e.g. `NFAST_KMLOCAL=\\<RFS IP address>\local` or `NFAST_KMLOCAL=\\<RFS Name>\local`.
   - Check that you can see the remote folder from Server 1 and Server 2 by running:

     ```
     dir "%NFAST_KMLOCAL%"
     ```

   - Ensure that all users granted permission to use the SQL Server and SQLEKM provider can see the remote folder in this way.
5. Set up the RFS to use the nShield Connect(s), and the nShield Connect(s) to use the RFS. See the *nShield Connect User Guide* for help.
6. Set up the nShield Connect(s) to use Server 1 and Server 2 as clients, and for the clients to use the nShield Connect(s). See the *nShield Connect User Guide* for help.
7. Create or load the desired Security World on the RFS or an nShield Connect. Ensure the Security World is loaded onto each nShield Connect used in the configuration. See

the *User Guide* for your HSM if you require help.

Before using the SQLEKM provider (see Using the SQLEKM provider), you should confirm that a Security World exists and that the HSM is usable: run `nfkminfo` and confirm that the World state returned shows `Usable` and that the desired `Module n` state (where `n` is the number of the module) shows `0x2 Usable`. For more information, see "Creating and managing a Security World" in the *User Guide* for your HSM. Then, create an OCS or softcard (see Security Worlds, key protection and failover recovery).

3.2.3. Failover cluster with HSMs in an AlwaysOn availability group

These procedures have been tested for an availability group that used two servers. Server 1 held a (nominal) primary replica, Server 2 held a (nominal) secondary replica. Primary and secondary replicas were read/write. The configuration used nShield Connect HSMs, and no shared disk. Each server could be logged into directly, or through a cluster availability group (virtual) address. The configuration also required a third server to act as RFS.

The procedures described here are based on this configuration.

3.2.3.1. Setting up and switching on TDE

The following steps should be performed for each database, the primary, and each secondary, that is part of the availability group, and for which you wish to switch on TDE encryption.

Before starting, it is assumed that the database you wish to encrypt:

- Already exists
- Is already part of an availability group within a cluster
- Is not currently encrypted, and includes no database encryption key (TDEDEK)
- Has never been encrypted before. If it has, you may see errors and a request or a log backup. In this case, refer to Taking a log backup.

In the examples shown here, the database to be encrypted is called SourceDatabase, and the database wrapping key is called `ekmWrappingKey` in the SQLEKM provider, and `dbWrappingKey` in the master database. Change names or other parameters to your own requirements. Also, these steps assume that a wrapping key of the same name does not already exist in either the SQLEKM provider or the master database.

The examples show T-SQL code options for using either an OCS or else a softcard credential. Select which option you prefer and maintain that choice throughout the examples (comment out the option you do not wish to use). In these examples the OCS

option is chosen.

Assuming that your servers and database(s) are already configured within an availability group, and you will use nShield Connects as your HSM modules, please prepare by making sure:

- You have SQL Server logins and appropriate permissions to configure or access the SQL Server and nShield software to be installed. This may include remote access authorization. If your SQL Server process is running as an autonomous service user, this must be granted appropriate permissions. You may need your system administrator to provide consent.
- Your nShield Security World and nShield Database Security Option Pack software is installed and configured in the same manner as that described in Failover cluster using nShield Connect XC HSMs (for this case, you may ignore the shared disk, as an availability group cluster can function without one).
- Your SQLEKM provider is enabled as described in Enabling the SQLEKM provider, you have created a suitable Security World on the RFS and which is loaded onto the nShield Connects. See the *nShield Connect User Guide* for help.
- You have created an OCS cardset, or softcard, as credential. Please refer to the *User Guide* for your HSM for further information about creating an OCS or a softcard. If you are using OCS cards, they must have a 1/N quorum, all be programmed with the exact same passphrase, and be from the same OCS cardset. We recommend a strong passphrase. Check your organization's security policies.
- If you are using OCS cards, you must have at least the same number (N) as HSMs you will be using. An OCS card must be inserted into the card reader of each HSM.
- The person managing or setting up the TDE encryption keys must use the same OCS or softcard for their login credential as is used for the tdeCredential below.

Before proceeding with the following steps:

- Make sure your database is recently backed up
- Make sure that primary and secondary replicas are synchronized within the availability group, and that failover can occur without any data loss
- If you prefer a particular server for the primary role, then you are failed over to that server
- You should also remember the roles (primary/secondary) that each server node starts with.

Perform the following steps in the order shown. The following description is written as if the server nodes retain the initial primary or secondary roles they begin with. You can use the availability group cluster virtual address, and manually failover between the nodes in order

to access them, but bear in mind this description refers to the initial (starting) role of each node, even if its actual role later changes.

1. On Primary: Set up the database wrapping key, TDE credential and login:

```
--Make sure you are running this on the PRIMARY.
--This script sets up a TDE wrapping key, login and credential on the primary.
--Create wrapping key
USE master
CREATE ASYMMETRIC KEY dbWrappingKey FROM PROVIDER SQLEKM
WITH PROVIDER_KEY_NAME='ekmWrappingKey',
CREATION_DISPOSITION = CREATE_NEW, ALGORITHM = RSA_2048;
GO
--Create wrapping key credential. Select one of OCS card, or else softcard.
--Comment out option you do not want to use.
--OCS card example
USE master
CREATE LOGIN tdeLogin FROM ASYMMETRIC KEY dbWrappingKey;
CREATE CREDENTIAL tdeCredential WITH IDENTITY = 'OCS1', SECRET = '+453X7V]MR'
FOR CRYPTOGRAPHIC PROVIDER SQLEKM;
ALTER LOGIN tdeLogin ADD CREDENTIAL tdeCredential;
GO
--Softcard example. Not used here, so commented out.
--Use master
--CREATE LOGIN tdeLogin FROM ASYMMETRIC KEY dbWrappingKey;
--CREATE CREDENTIAL tdeCredential WITH IDENTITY = 'scard1', SECRET = '0O*dG0ffz2'

--FOR CRYPTOGRAPHIC PROVIDER SQLEKM;
--ALTER LOGIN tdeLogin ADD CREDENTIAL tdeCredential;
```

2. On (each) secondary: Restart the SQL Server instance. Set up the database wrapping key, TDE credential and login.

```
--Make sure you are running this on the SECONDARY.
--NOTE the wrapping key must already exist, as created by the primary.
--This script opens a wrapping key, TDE login and credential on a secondary.
--The credential must match (same OCS cardset/softcard and password) as primary.
--Create wrapping key
USE master
CREATE ASYMMETRIC KEY dbWrappingKey FROM PROVIDER SQLEKM
WITH PROVIDER_KEY_NAME='ekmWrappingKey',
CREATION_DISPOSITION = OPEN_EXISTING; --Wrapping key should already have been created on the
primary.
GO
--Create wrapping key credential. Select one of OCS card, or else softcard.
--Comment out option you do not want to use.
--OCS card example
USE master
CREATE LOGIN tdeLogin FROM ASYMMETRIC KEY dbWrappingKey;
CREATE CREDENTIAL tdeCredential WITH IDENTITY = 'OCS1', SECRET = '+453X7V]MR'
FOR CRYPTOGRAPHIC PROVIDER SQLEKM;
ALTER LOGIN tdeLogin ADD CREDENTIAL tdeCredential;
GO
--Softcard example. Not used here, so commented out.
--Use master
--CREATE LOGIN tdeLogin FROM ASYMMETRIC KEY dbWrappingKey;
--CREATE CREDENTIAL tdeCredential WITH IDENTITY = 'scard1', SECRET = '0O*dG0ffz2'
--FOR CRYPTOGRAPHIC PROVIDER SQLEKM;
--ALTER LOGIN tdeLogin ADD CREDENTIAL tdeCredential;
```

3. On both primary and secondary, check the database remains synchronized. To do this, in SQL Server Management Studio, look at `Server name > Name of your database`. If after the previous steps you find that the database is now 'Not Synchronized', resynchronize by running the following query:

```
--Run on primary/secondary that appears to be unsynchronized with availability group.
USE master;
GO

ALTER DATABASE [SourceDatabase] SET HADR RESUME
```

   If the database remains unsynchronized after performing this step, you may have configuration problems. Attempt to correct this before proceeding.

4. On primary: Create the database encryption key and switch on TDE encryption.

```
--Make sure you are running this on PRIMARY
--Create actual database encryption key (TDEDEK)
USE SourceDatabase;
CREATE DATABASE ENCRYPTION KEY WITH ALGORITHM = AES_256
ENCRYPTION BY SERVER ASYMMETRIC KEY dbWrappingKey;
GO
--A short delay may be required here before switching on encryption.
WAITFOR DELAY '00:00:05'; -- Set delay period as required. One second = '00:00:01'
-- Break any connection with the SourceDatabase so that encryption can commence.
USE [master];
GO
-- Enable TDE (switch on encryption) on the SourceDatabase:
ALTER DATABASE SourceDatabase SET ENCRYPTION ON;
GO
```

   If your database has previously been encrypted you may see errors at this point. If you are asked to take a pending log backup please perform the query shown in Taking a log backup. Then, repeat the following:

```
-- Break any connection with the SourceDatabase so that encryption can commence.
USE [master];
GO
-- Enable TDE (switch on encryption) on the SourceDatabase:
ALTER DATABASE SourceDatabase SET ENCRYPTION ON;
GO
```

5. After performing the above steps, check the TDE encryption is switched on and the database is functioning correctly.

   First check the encryption state on the primary by running the following Encryption state check query:

```
-- Encryption state check. Returns the encryption state of databases.
SELECT DB_NAME(e.database_id) AS DatabaseName, e.database_id, e.encryption_state, CASE e.encryption_
state
```

```
WHEN 0 THEN 'No database encryption key present, no encryption'
WHEN 1 THEN 'Unencrypted'
WHEN 2 THEN 'Encryption in progress'
WHEN 3 THEN 'Encrypted'
WHEN 4 THEN 'Key change in progress'
WHEN 5 THEN 'Decryption in progress'
END AS encryption_state_desc, c.name, e.percent_complete FROM sys.dm_database_encryption_keys
AS e
LEFT JOIN master.sys.certificates AS c ON e.encryptor_thumbprint = c.thumbprint
```

The encryption state for your database should be marked as Encrypted (if it is marked as Encryption in progress, wait a while and try again).

You should now be able to failover to a secondary with no data loss. After failing over to the secondary, run the same query above to check the encryption state for your database is also Encrypted on the secondary.

Failover between the nodes in your configuration, and attempt some database queries while connected to each. Add data to the database, query that same data, then, delete the data you just added, or whatever other queries you think appropriate.

Satisfy yourself that all is functioning correctly before continuing to use the TDE encrypted database.

3.2.3.2. Taking a log backup

If you get an error requesting that you take a log backup, try adapting the following code to your own requirements, and then run it.

```
USE master;
GO
ALTER DATABASE <Name-of-your-database>
SET RECOVERY FULL;
GO
USE master;
GO
-- Note. You should have provided a path to your backups when setting up your
-- availability group.
EXEC sp_addumpdevice 'disk', '<Name-of-your-device>',
'<Path-to-your-backups>\<Name-of-your-log-backup-file>';
GO
-- Back up the log
BACKUP LOG <Name-of-your-database> TO <Name-of-your-device>;
GO
--Drop backup device
EXEC sp_dropdevice '<Name-of-your-device>';
```

Example:

```
USE master;
GO
ALTER DATABASE SourceDatabase
SET RECOVERY FULL;
```

```
GO

USE master;
GO
EXEC sp_addumpdevice 'disk', 'EncryptedSourceDatabaseBackupLog',
'\\Server-2\NetWorkShareFolder\SourceDatabase_20160210122459';
GO
-- Back up the log
BACKUP LOG SourceDatabase TO EncryptedSourceDatabaseBackupLog;
GO
--Drop backup device
EXEC sp_dropdevice 'EncryptedSourceDatabaseBackupLog';
```

3.2.3.3. Removing TDE encryption from an AlwaysOn availability group

This procedure assumes you have already successfully set up TDE encryption in a similar manner to that described in Setting up and switching on TDE.

Perform the following steps in the order shown.

1. On primary: Switch off TDE encryption.

```
--Run this on PRIMARY in high availability group environment.
--Switch off TDE encryption.
USE [master];
ALTER DATABASE SourceDatabase SET ENCRYPTION OFF;
GO
```

2. On primary: Wait until decryption has finished. Check this by using the Encryption state check. When decryption has completed, continue to next step.
3. On primary: Drop the database encryption key (TDEDEK).

```
--Drop the database encryption key (TDEDEK)
USE SourceDatabase
DROP DATABASE ENCRYPTION KEY;
```

4. On (each) secondary: Drop TDE login and credential, and wrapping key (TDEKEK) from database.

```
--You must have switched off TDE encryption on primary before running this script.
--Run this on SECONDARY in high availability group environment.
USE master;
GO
--Drop the TDE credential and login
ALTER LOGIN tdeLogin DROP CREDENTIAL tdeCredential;
DROP LOGIN tdeLogin;
DROP CREDENTIAL tdeCredential;

--Drop the wrapping key from database only
DROP ASYMMETRIC KEY dbWrappingKey;
```

5. On primary: Drop TDE login and credential, and wrapping key (TDEKEK) from database. If you also wish to drop the wrapping key (TDEKEK) from the SQLEKM provider, be sure it is safe to do so.

```
--Run this on PRIMARY in high availability group environment.
USE master;
GO
--Drop the TDE credential and login on primary.
ALTER LOGIN tdeLogin DROP CREDENTIAL tdeCredential;
DROP LOGIN tdeLogin;
DROP CREDENTIAL tdeCredential;
--Select option below to remove wrapping key from database only, or both database
--and SQLEKM provider.
--If you remove the wrapping key copy from the SQLEKM provider, it will be lost
--forever. If you do this, be sure this is what you want to do.
--Drop the wrapping key from database only
DROP ASYMMETRIC KEY dbWrappingKey;
--Drop the wrapping key from both database and SQLEKM provider
--DROP ASYMMETRIC KEY dbWrappingKey REMOVE PROVIDER KEY;
```

6. After performing the above steps, check the TDE encryption is switched off on the primary by running the same Encryption state check query as shown above. The previously encrypted database should no longer be listed.

   You may see the tempdb database remains shown as Encrypted. To remove this, restart the SQL Server instance.

   Failover to a secondary, and check that there is no data loss. Run the same Encryption state check query on the secondary as shown above. The previously encrypted database should no longer be listed.

3.3. Security Worlds, key protection and failover recovery

This section highlights some considerations when choosing Security World and key protection options for use with the SQLEKM provider. It focusses on recovery of Security World authorization where a system has temporarily failed (for instance after a power outage) and is then returned to operation. This does not apply to other failure recovery functions. These considerations are applicable to both standalone systems and database failover clusters. For a fuller explanation of Security Worlds and key protection please refer to the *User Guide* for your HSM.

Module protected keys are not supported by the SQLEKM provider. Therefore, direct protection of encryption keys that can be used without requiring further authorization mechanisms is not possible.

In the event of a temporary failure of the SQLEKM provider, there may be a consequent loss of:

- Credential authorization.
- FIPS authorization (only if using a FIPS 140-2 Level 3 Security World).

A credential authorization can be granted using either a softcard or an OCS card, with passphrase. In the case of an OCS, a card must be always available in a valid HSM card reader in order to grant reauthorization after a failure, and permit automatic recovery. See Creating a credential for more information.

Where FIPS authorization is required, this can be granted either by using an operator card specifically for this purpose, or through an operator card that is also used for credential authorization. A card from the OCS must be always available in a valid HSM card reader in order to grant re-authorization after a failure, and permit automatic recovery.

Never use ACS cards for FIPS authorization, as they will not support automatic recovery.

Using these options, a summary of the authorization recovery behavior of the SQLEKM provider after a temporary outage is given in the table below.

Security World type	Protection / Credential	Standalone system	Database cluster
FIPS 140-2 Level 2 CC-CMTS	Softcard	Recovers automatically.	Recovers automatically.
	OCS	Use OCS for credential authorization: • Use 1/N quorum. Same passphrase for all cards. • Leave an operator card in HSM slot. Recovers automatically.	Use OCS for credential authorization: • Use 1/N quorum. Same passphrase for all cards. • Leave an operator card in slot of every HSM in cluster. Recovers automatically.

Security World type	Protection / Credential	Standalone system	Database cluster
FIPS 140-2 Level 3	Softcard	Use OCS for FIPS authorization (only): • Leave an operator card in HSM slot. Recovers automatically.	Use OCS for FIPS authorization (only): • Leave an operator card in slot of every HSM in cluster. Recovers automatically.
	OCS	Use OCS for both credential and FIPS authorization: • Use 1/N quorum. Same passphrase for all cards. • Leave an operator card in HSM slot. Recovers automatically.	Use OCS for both credential and FIPS authorization: • Use 1/N quorum. Same passphrase for all cards. • Leave an operator card in slot of every HSM in cluster. Recovers automatically.

If you are using an OCS to facilitate automatic recovery of the SQLEKM provider:

- If you are using the OCS for credential authorization, all must be members of the same cardset for the same credential, and the same passphrase must be assigned to every card in the set.
- If you are using the OCS for FIPS authorization purposes only, the quorum automatically defaults to 1/N, and (any) passphrase is ignored.

Authorization acquired through a persistent operator card will not automatically reinstate itself after loss due to a temporary failure.

4. Using the SQLEKM provider

This section shows you how to enable the SQLEKM provider, and provides examples showing how to encrypt and decrypt data.

To run these examples, open SQL Server Management Studio and connect to a SQL Server instance, then open a query window to execute a query. In the example T-SQL statements, the names used for cryptographic keys (such as `dbAES256Key`) and databases (such as `TestDatabase`) are example names only. The exception to this rule is the `master` database, which is a real database.

If you are using a failover cluster, run the examples through the virtual server. Otherwise, use the active server in the cluster. Note that any directory/file paths will be relative to the active server.

You must have a SQL Server login and appropriate permissions to configure or access the SQL Server or SQLEKM provider.

If a database is started before the SQLEKM provider is ready to accept a connection, the database might enter a `pending recovery` state. By default, 20 retries of provider initialization are attempted, with each attempt occurring at 5-second intervals. If necessary, the number of retry attempts can be configured by adding the following registry entry (type `REG_DWORD`):

```
HKLM\SOFTWARE\nCipher\SQLEKM\NCoreMaxInitRetries
```

4.1. Enabling the SQLEKM provider

1. To enable the SQLEKM provider, execute the following query:

```
sp_configure 'show advanced options', 1; RECONFIGURE;
GO
sp_configure 'EKM provider enabled', 1; RECONFIGURE;
GO
```

2. Register the SQLEKM provider by executing the following query:

```
CREATE CRYPTOGRAPHIC PROVIDER <Name of provider>
FROM FILE = '<Path to provider>';
GO
```

Where:

- *<Name of provider>* is the name that is used to refer to the SQLEKM provider, e.g. SQLEKM.
- *<Path to provider>* is the fully qualified path to the `ncsqlekm.dll` file, e.g. `C:\Program Files\nCipher\nfast\bin\ncsqlekm.dll`.

3. To check that the SQLEKM provider is listed:
   a. Open SQL Server Management Studio on the Management Studio.
   b. Go to **Security > Cryptographic Providers**. You should see ***<Name of provider>***.

4.2. Creating a credential

A SQL Server credential represents the OCS, or softcard, and associated passphrase that is used to authorize access to specific keys protected by the Security World. The OCS or softcard must already exist before attempting to create a credential. When using an OCS cardset with the SQLEKM provider, use a 1/N quorum.

Encryption keys can be protected by only one OCS cardset, or else softcard, at any one time. By implication, this also applies to the SQL Server credential that represents that OCS cardset or softcard.

You can transfer key(s) from one OCS cardset to another OCS cardset, or from one softcard to another softcard. You must use the `rocs` utility to perform the key transfer. Please see the *User Guide* for your HSM for more details. However, you cannot transfer keys between an OCS cardset and softcard.

If you are using a failover cluster, you will need to create the OCS or softcard directly through the active server. Please refer to the *User Guide* for your HSM for further information about creating an OCS or a softcard.

We recommend the use of a strong passphrase. Please consult your organization's security policies.

Once created, the credential must in turn be associated with a particular login before it can be used. The owner of that login is then authorized to use that credential to create or use encryption keys that are protected by the OCS or softcard related to the credential.

A login can be associated with only one credential at a time, but a credential can be associated with several logins at a time.

It is by use of credentials and logins that access to encryption keys for use in SQL Server can be controlled through the SQLEKM provider. For this reason, you should restrict who

can use a credential. It is beyond the scope of this guide to deal with user access permissions. However, please be aware that if a valid credential and associated OCS card or softcard is available to an unauthorized user, who is then able to associate that credential with their login, this represents a security risk (the token's password is stored in the credential and cannot be used to identify the user). This may be less of an issue when using TDE encryption, for which users authorized to access the database do not need an associated credential in any case, but it may be an issue with Cell encryption.

Countermeasures to reduce these risks may be made through SQL Server or Windows access permissions in accordance with your security policies. Options that may be considered are to restrict use of the OCS or softcards by identifying the relevant files amongst the Security World data, and setting their access permissions to authorized users only. You can identify OCS cards and softcards using the `nfkminfo` utility as follows:

- OCS cards: use `nfkminfo -c`
- Softcards: use `nfkminfo -s`.

You will see the OCS card or softcard names and their associated hash number. Look in the Security World data and set appropriate permissions for all files that share the same hash number as the OCS or softcard you are protecting, see Security World data and back-up and restore for more information about file hash numbers.

You may use multiple credentials if you wish to simultaneously use TDE and cell-level encryption. You are advised to set up your cell-level credentials and associated encryption keys first, before setting up the TDE login/credential and switching TDE on, see Transparent Data Encryption - TDE and Cell Level Encryption (CLE).

To create a credential and map it to a login:

1. In SQL Server Management Studio, navigate to **Security > Credentials**.
2. Right-click **Credentials**, then select **New Credential**.
3. Set **Credential name** to ***loginCredential***.
4. Set **Identity** to *<OCSname>*, where *<OCSname>* matches the name of the OCS or softcard. You must match the character case.
5. Set **Password** to *<passphrase>*, where *<passphrase>* matches the passphrase on the card set or softcard. You must match the character case.
6. Ensure **Use Encryption Provider** is selected, then from the ***<Name of provider>***, drop-down list, choose ***<Name of provider>***. Click **OK**.
7. Check that under **Security > Credentials** the name of the new credential appears. If necessary, right click and select **Refresh**.

8. In **SQL Server Management Studio**, navigate to **Security > Logins**.
9. Right-click to select the required login, then select **Properties**.
10. Ensure **Map to Credential** is selected, then select **loginCredential** from the drop down list. Click **Add**, then click **OK**.

4.3. Checking the configuration

To check that the SQLEKM provider was configured correctly:

1. Check that the SQLEKM provider was registered correctly by running the following query:

   ```
   SELECT * FROM sys.cryptographic_providers;
   ```

   A table is displayed with information about the registration of the SQLEKM provider. Check that:

   - The build version matches the `sqlekm` version number (found in the SQLEKM installation versions file).
   - The `.dll` path matches the path given when registering the SQLEKM provider (e.g. `C:\Program Files\nCipher\nfast\bin\ncsqlekm.dll`.)
   - The `is_enabled` column is set to `1`.
2. Check the SQLEKM provider properties by running the following query:

   ```
   SELECT * FROM sys.dm_cryptographic_provider_properties;
   ```

   A table is displayed with information about the properties of the SQLEKM provider. Check that:

   - `provider_version` matches the `sqlekm` version number (found in the SQLEKM installation versions file). The number may be in a different format, but digits should be the same.
   - `friendly_name` is `nCipher SQLEKM Provider`
   - `authentication_type` is set to `BASIC`
   - `symmetric_key_support` is set to `1`
   - `asymmetric_key_support` is set to `1`
3. To check that the supported cryptographic algorithms can be queried, run the following query:

   ```
   DECLARE @ProviderId int;
   ```

```
SET @ProviderId = (SELECT TOP(1) provider_id FROM sys.dm_cryptographic_provider_properties
WHERE friendly_name LIKE 'nCipher SQLEKM Provider');
SELECT * FROM sys.dm_cryptographic_provider_algorithms(@ProviderId);
GO
```

A table is displayed with the supported cryptographic algorithms. For more information about the algorithms that should be displayed, see Supported cryptographic algorithms.

4.4. Encryption and encryption keys

When you have configured the SQLEKM provider, and you have a suitable credential associated with your login, you can use the SQLEKM provider to:

- Manage cryptographic keys within the nShield HSM.
- Encrypt or decrypt entire databases or fields within tables within your SQL Server service using TDE or Cell encryption, or both at the same time.

Encryption keys can be created in the SQLEKM provider and referenced by the appropriate database as required for use. When a reference of an encryption key is no longer required for active use in the database, it should be deleted from the database while retaining the original copy of the key in the SQLEKM provider, which also acts as a secure backup. Storing original copies of encryption keys in the SQLEKM provider is more secure than leaving encryption key references and associated data together in the database. So long as you retain a copy of the original key in the SQLEKM provider, its reference can be restored when next required for active use in the database.

Copying and deletion of keys does not apply to a TDE Database Encryption Key (TDEDEK), which is created as an integral part of a user database. On the other hand, this can apply to the wrapping key (TDEKEK) which is used to protect the TDEDEK. See Transparent Data Encryption - TDE.

Copies of encryption keys that are retained in the SQLEKM provider (or Security World) are in turn protected by inbuilt encryption facilities, and cannot be read or decrypted without suitable authorization mechanisms. Even if a Security World or HSM is stolen, it will be useless to anyone who does not have access to the correct authorization mechanisms.

You must be very careful if you consider deleting an original encryption key from the SQLEKM provider; once deleted from there, it is lost for good, unless you have a prior backup of the Security World. Similarly, you must be very careful before dropping any of the authorization mechanisms such as OCS cards, softcards, ACS cards, and their associated passwords.Loss of these could also mean you lose access to your encryption

keys.

It is recommended to regularly re-encrypt your data using fresh encryption keys so that any persistent attempts to decipher or compromise your encrypted data are impeded.

Encryption keys that have been made accessible to a database through the SQLEKM provider are accessible through references provided to the database. Copies of the real keys do not exist in the database.

4.5. Key naming, tracking and other identity issues

Encryption keys held in the database are really references to actual keys held in SQLEKM provider. For the purpose of key tracking, it is suggested that you use the same name for both the database and SQLEKM provider version of an encryption key. Use a suffix or prefix to distinguish between the database and SQLEKM provider versions.

In a database there can be only one key with a specific name at any one time. However, note that key names can be duplicated for different keys in the SQLEKM provider.

Even though possible, we strongly discourage permitting duplicate key names in the SQLEKM provider, since this simply leads to confusion and potential operational errors.

If you have very many keys, you may wish to implement a key naming convention that helps you track which keys encrypt which data, backed up with some form of secure documentation. Note if a key naming convention incorporates a database identifier, a Security World can hold keys for more than one database at the same time, and a key can be used in more than one database at a time.

If you are using more than one Security World you should ensure you can physically identify the ACS and OCS cards that belong to each Security World.

Once a Security World is loaded onto a HSM, its OCS cards can be inserted into the card reader and individually identified with cardset name and creation sequence number using Entrust supplied utilities.

Additionally, you can name individual OCS cards when the OCS cardset is created. The keys a card is protecting can be identified using the `rocs` utility.

To use the examples in this document you will first need to create `TestDatabase` and `TestTable` as shown in Creating a database and Creating a table. Otherwise, provide your own database and table to perform encryption operations and adapt the examples accordingly. Refer to Verifying by inspection that TDE has occurred on disk before adapting any examples. See also T-SQL shortcuts and tips.

Encryption keys created under a login that is mapped to a particular credential will be protected by that credential. If you wish to transfer keys to another OCS or softcard please see the *User Guide* for your HSM.

You can check which keys are protected under which credential by using the `rocs` utility; see the *User Guide* for your HSM for details. If you are using `rocs` in a failover cluster environment, you must use it on the active server.

If you are protecting encryption keys with an OCS credential, an operator card must be inserted into the HSM card reader of every HSM that is part of the configuration to create or authorize use of the encryption keys.

4.6. Supported cryptographic algorithms

The algorithms that you can use for encryption depends on the type of Security World being used.

The following table lists cryptographic algorithms that are available when using symmetric keys.

Algorithm	FIPS 140-2 Level 2 Security World	FIPS 140-2 Level 3 v1/v2 Security World	FIPS 140-2 Level 3 v3 Security World	CC-CMTS Security World
DES	Yes	No	No	No
Triple_DES	Yes	Yes	No	No
Triple_DES_3KEY	Yes	Yes	No	No
AES_128	Yes	Yes	Yes	Yes
AES_192	Yes	Yes	Yes	Yes
AES_256	Yes	Yes	Yes	Yes

The following table lists cryptographic algorithms that are available when using asymmetric cryptographic keys.

Algorithm	FIPS 140-2 Level 2 Security World	FIPS 140-2 Level 3 v1/v2 Security World	FIPS 140-2 Level 3 v3 Security World	CC-CMTS Security World
RSA_512	Yes	Yes	No	No
RSA_1024	Yes	Yes	No	No
RSA_2048	Yes	Yes	Yes	Yes
RSA_3072	Yes	Yes	Yes	Yes
RSA_4096	Yes	Yes	Yes	Yes

Although DES and RSA_512 keys can be used, this is mainly for compatibility with legacy systems. Otherwise they are not recommended for use with nShield products. For more information, contact Entrust nShield Technical Support.

4.6.1. Symmetric keys

4.6.1.1. Symmetric key GUIDs

When a new symmetric key is generated through the SQLEKM provider, it is associated in the database with a *Global Unique Identifier* or GUID. The database issues a different and random GUID for every new key, and uses the GUID to identify the correct symmetric key for encryption or decryption purposes. As long as a copy of this key with the same GUID remains available to the database, it can be used indefinitely.

If the key is lost to the database, then a cryptographically equivalent duplicate can be generated through the SQLEKM provider from the copy stored in the HSM. The duplicate key, although cryptographically identical to the lost key, will be issued with a new GUID by the database. Because the GUID is different from the original key it will not be identified with the original key, and will not be allowed to perform encryption or decryption of the data with which the lost key was associated.

To avoid this issue, you should always specify an `IDENTITY_VALUE` when generating a symmetric key. `IDENTITY_VALUE` is used to generate the key GUID in the database. The examples below create a symmetric key in the SQLEKM provider, and make available the same key for use in the database. The key does not have to share the same name between the SQLEKM provider and database.

4.6.1.2. Original key

To create a symmetric key with an identity value:

```
USE <Your_database_name>
CREATE SYMMETRIC KEY <Name_of_key_in_database> FROM PROVIDER <Name_of_SQLEKM_provider>
WITH PROVIDER_KEY_NAME='<Name_of_Key_in_SQLEKM_provider>',
IDENTITY_VALUE='<Unique_GUID_generator_string>',
CREATION_DISPOSITION = CREATE_NEW, ALGORITHM=<Symmetric_algorithm_desc>;
GO
```

Where

- *<Your_database_name>* is the name of the database for which you wish to provide encryption. See T-SQL shortcuts and tips for examples.
- *<Name_of_SQLEKM_provider>* is the name of the SQLKM provider you are using.
- *<Name_of_key_in_database>* is the name you wish to give the key in the database.
- *<Name_of_key_in_SQLEKM_provider>* is the name you wish to give the key in the SQLEKM provider. Please note that there is a length restriction on this name of 31 characters maximum if created using a T-SQL query.
- *<Unique_GUID_generator_string>* is a unique string that will be used to generate the GUID.
- *<Symmetric_algorithm_desc>* is a valid symmetric key algorithm descriptor.

If the value of the *<Unique_GUID_generator_string>* is known to an attacker, this will help them reproduce the symmetric key. Therefore, it should always be kept secret and stored in a secure place. We recommend the *<Unique_GUID_generator_string>* shares qualities similar to a strong passphrase. Check your organization's security policy.

Only one key that has been created using a particular `IDENTITY_VALUE` can exist at the same time in the same database.

4.6.1.2.1. Creating a duplicate key

This example shows how a duplicate of a lost symmetric key can be made through the SQLEKM provider from the HSM copy, and imported into the database.

To create a duplicate key:

```
USE <Your_database_name>
CREATE SYMMETRIC KEY <Name_of_key_in_database> FROM PROVIDER <Name_of_SQLEKM_provider>
WITH PROVIDER_KEY_NAME='<Name_of_Key_in_SQLEKM_provider>',
IDENTITY_VALUE='<Unique_GUID_generator_string>',
CREATION_DISPOSITION = OPEN_EXISTING;
GO
```

Where *<Unique_GUID_generator_string>* is the same value as used to create the original key.

4.6.1.3. Creating and managing symmetric keys

If you are using a credential based on an OCS, ensure that your operator card is inserted in the HSM card reader before attempting to create and manage symmetric keys.

This query generates a new symmetric key through the SQLEKM provider which will be protected inside the HSM. It then makes the key available to the database.

```
USE TestDatabase
CREATE SYMMETRIC KEY dbAES256Key
FROM PROVIDER <Name of SQLEKM provider>
WITH PROVIDER_KEY_NAME='ekmAES256Key',
IDENTITY_VALUE='Rg7n*9mnf29xl4',
CREATION_DISPOSITION = CREATE_NEW, ALGORITHM=AES_256;
GO
```

Where *<Name of SQLEKM provider>* is the name that is used to refer to the SQLEKM provider.

In this example, the key is named `dbAES256Key` in the database and `ekmAES256Key` in the SQLEKM provider.

4.6.1.4. Listing symmetric keys in a database

To list the symmetric keys in a database:

1. Open SQL Server Management Studio on the Management Studio.
2. Go to **Databases > TestDatabase > Security > Symmetric Keys** (right-click to select **Refresh**).

Alternatively, you may check keys by following the methods shown in Checking keys.

4.6.1.5. Removing symmetric keys from the database only

To remove the symmetric key `dbAES256Key` from the database only (`TestDatabase`):

```
USE TestDatabase
DROP SYMMETRIC KEY dbAES256Key;
GO
```

After the above query completes, the key dbAES256Key is deleted from the database, but the corresponding key ekmAES256Key remains in the HSM and is accessible through the SQLEKM provider.

4.6.1.6. Re-importing symmetric keys

To re-import the symmetric key (dbAES256Key) that was removed from the database, where a corresponding copy (ekmAES256Key) exists in the HSM:

```
USE TestDatabase
CREATE SYMMETRIC KEY dbAES256Key FROM PROVIDER <Name of provider>
WITH PROVIDER_KEY_NAME='ekmAES256Key',
IDENTITY_VALUE='Rg7n*9mnf29x14',
CREATION_DISPOSITION = OPEN_EXISTING;
GO
```

This example uses the same IDENTITY_VALUE as in the original key generation. This regenerates the same GUID. Having the same GUID means that the key is logically identical to the key it replaces.

4.6.1.7. Removing symmetric keys from the database and provider

To remove a symmetric key (dbAES256Key) from both the database (TestDatabase) and the nShield HSM, execute the following query:

```
USE TestDatabase
DROP SYMMETRIC KEY dbAES256Key REMOVE PROVIDER KEY;
GO
```

Using this method means you do not have to name the corresponding key in the SQLEKM provider to remove it from there.

Refer to your security policies before considering deleting a SQLEKM provider key from the HSM. You cannot import a key into the database once you have deleted that key from the SQLEKM provider. Once deleted from the SQLEKM provider, if you have no Security World backup copy of that key, it will be lost.

4.6.2. Creating and managing asymmetric keys

If you are using a credential based on an OCS, ensure that your operator card is inserted in the HSM card reader before attempting to create and manage asymmetric keys.

4.6.2.1. Creating an asymmetric key

The following query generates a new asymmetric key in the SQLEKM provider which will be protected inside the HSM, and then makes the key available to the database:

```
USE TestDatabase
CREATE ASYMMETRIC KEY dbRSA2048Key FROM PROVIDER <Name_of_key_in_SQLEKM_provider>
WITH PROVIDER_KEY_NAME='ekmRSA2048Key',
CREATION_DISPOSITION = CREATE_NEW, ALGORITHM=RSA_2048;
GO
```

<Name_of_key_in_SQLEKM_provider> is the name you wish to give the key in the SQLEKM provider. Please note that there is a length restriction on this name of 31 characters maximum if created using a T-SQL query.

This example names the key `dbRSA2048Key` in the database, and `ekmRSA2048Key` in the SQLEKM provider.

`IDENTITY_VALUE` is not a supported argument for asymmetric key generation.

4.6.2.2. Listing asymmetric keys in a database

To list the asymmetric keys in a database:

1. Open SQL Server Management Studio on the Management Studio.
2. Go to **Databases > TestDatabase > Security > Asymmetric Keys** (right-click to select **Refresh**).

Alternatively, you may check keys by following the methods shown in Checking keys.

4.6.2.3. Removing an asymmetric key from the database only

To remove the asymmetric key `dbRSA2048Key` from the database only (`TestDatabase`):

```
USE TestDatabase
DROP ASYMMETRIC KEY dbRSA2048Key;
GO
```

After the above query completes, the key `dbRSA2048Key` is deleted from the database, but the corresponding key `ekmRSA2048Key` remains in the SQLEKM provider.

4.6.2.4. Re-importing an asymmetric key

To re-import a deleted asymmetric key (dbRSA2048Key) back into the database (TestDatabase), where a corresponding copy (ekmRSA2048Key) exists in the SQLEKM provider:

```
USE TestDatabase
CREATE ASYMMETRIC KEY dbRSA2048Key
FROM PROVIDER <Name of provider> WITH PROVIDER_KEY_NAME='ekmRSA2048Key',
CREATION_DISPOSITION = OPEN_EXISTING;
GO
```

4.6.2.5. Removing an asymmetric key from the database and provider

To remove the asymmetric key (dbAES256Key) from both the database (TestDatabase) and the nShield HSM, execute the following query:

```
USE TestDatabase
DROP ASYMMETRIC KEY dbRSA2048Key REMOVE PROVIDER KEY;
GO
```

Using this method means you do not have to name the corresponding key in the SQLEKM provider to remove it from there.

Refer to your security policies before considering deleting a SQLEKM provider key from the HSM. You cannot import a key into the database once you have deleted that key from the SQLEKM provider. Once deleted from the SQLEKM provider, if you have no Security World backup copy of that key, it will be lost.

4.6.2.6. Creating a symmetric wrapped key from an asymmetric wrapping key

To create a symmetric wrapped key (dbSymWrappedKey1) from an asymmetric wrapping key (dbAsymWrappingKey1), execute the following query:

```
USE TestDatabase
CREATE ASYMMETRIC KEY dbAsymWrappingKey1 FROM PROVIDER <Name of provider>
WITH PROVIDER_KEY_NAME='ekmAsymWrappingKey1',
CREATION_DISPOSITION = CREATE_NEW, ALGORITHM=RSA_2048;
CREATE SYMMETRIC KEY dbSymWrappedKey1
WITH ALGORITHM = AES_128,
IDENTITY_VALUE ='yr7s365$dfFJ901'
ENCRYPTION BY ASYMMETRIC KEY dbAsymWrappingKey1;
```

Where *<Name of provider>* is the name that is used to refer to the SQLEKM provider.

If you wish to delete the wrapped and wrapping keys, you will have to delete the wrapped key first.

4.6.3. Importing keys

By 'importing keys' we should distinguish between:

- Importing a key into the database that was created in the SQLEKM provider.
- Importing a (foreign) key that was created outside the SQLEKM provider into its Security World.

Keys created in the SQLEKM provider can be imported into a database provided they are in simple format.

As regards keys created outside the SQLEKM provider, it is not recommended to import such keys into the Security World unless they are from a trustworthy source. Importing of externally created keys into the Security World may require format conversion. Entrust provides limited off the shelf key import facilities through use of the `generatekey` utility or `KeySafe` application (no key export facilities are supplied).

Please contact Entrust nShield Technical Support if you wish to pursue key import (or export) operations further.

To import an externally created symmetric key with an identity value:

```
USE <Your_database_name>
CREATE SYMMETRIC KEY <Name_of_key_in_database> FROM PROVIDER
<Name_of_SQLEKM_provider>
WITH PROVIDER_KEY_NAME='<Name_of_Key_in_SQLEKM_provider>',
IDENTITY_VALUE='<Unique_GUID_generator_string>',
CREATION_DISPOSITION = OPEN_EXISTING;
```

Where:

- *<Your_database_name>* is the name of the database for which you wish to provide encryption. See T-SQL shortcuts and tips for examples.
- *<Name_of_SQLEKM_provider>* is the name of the SQLKM provider you are using.
- *<Name_of_key_in_database>* is the name you wish to give the key in the database.
- *<Name_of_key_in_SQLEKM_provider>* is the name of the externally created key in the SQLEKM provider. This must be no more than 32 characters maximum.
- *<Unique_GUID_generator_string>* is a unique string that will be used to generate the GUID.

If the value of the *<Unique_GUID_generator_string>* is known to an attacker, this will help them reproduce the symmetric key. Therefore, it should always be kept secret and stored in a secure place. We recommend that the *<Unique_GUID_generator_string>* shares qualities similar to a strong passphrase. Check your organization's security policy.

Only one key that has been created using a particular `IDENTITY_VALUE` can exist at the same time in the same database.

To import an externally created asymmetric key

```
USE <Your_database_name>CREATE ASYMMETRIC KEY <Name_of_key_in_database> FROM PROVIDER<Name_
of_SQLEKM_provider>
WITH PROVIDER_KEY_NAME='<Name_of_Key_in_SQLEKM_provider>', CREATION_DISPOSITION = CREATION_
DISPOSITION = OPEN_EXISTING;
```

Parameters are the same as for the symmetric key. Note, for an externally created asymmetric key, name length restriction of 32 characters maximum applies for *<Name_of_key_in_SQLEKM_provider>*

4.7. Transparent Data Encryption - TDE

An example of configuring an AlwaysOn availability group with no shared disk for TDE encryption is given in Failover cluster with HSMs in an AlwaysOn availability group.

These examples assume that both the `TestDatabase` and `TestTable` as described in T-SQL shortcuts and tips have been created, and are not currently encrypted.

When TDE encryption has been correctly set up and switched on, the database it is protecting will appear as normal to any user who has been granted suitable permissions to use the database. The user does not require any SQLEKM provider credential to access or modify TDE protected data.

Note that:

- If the credential protecting the TDE encryption key is OCS based, the operator cards must be inserted in the HSM card reader for the TDE encryption to be set up and authorized.
- The person setting up or managing the TDE encryption keys must use the same OCS or softcard for their login credential as used for the `tdeCredential` below.

The TDE Database Encryption Key (TDEDEK) is a symmetric key that is used to perform the actual encryption of the database. It is created by SQL Server and cannot be exported from the database meaning that it cannot be created or directly protected by the SQLEKM provider. In order to protect the TDEDEK within the database it may in turn be encrypted by a wrapping key. The wrapping key is called the TDE Key Encryption Key (TDEKEK). In this case, the SQLEKM provider can create and protect the TDEKEK.

Before running the following examples, you should create a backup copy of the unencrypted database. See Backing up a database with SQL Server Management studio. Alternatively, you may prefer to adapt the T-SQL query shown in Making a database backup. Save the backup as `<Drive>:\<Backup_directory_path>\TestDatabase_TDE_Unencrypted.bak`.

If you are using a shared disk cluster as described earlier in this document, then to set up TDE encryption, it should normally be sufficient to perform the following steps on the active node only:

- Create TDEKEK
- Set up TDE login and credential
- Create TDEDEK and switch on encryption.

These steps are described in more detail below. If these steps are performed on the active node, then the TDE set up should be automatically inherited when you failover to the other node. You should not have to repeat the TDE set up on the second node. This does not apply if you are using an AlwaysOn availability group with no shared disk. In this case, please see Failover cluster with HSMs in an AlwaysOn availability group.

4.7.1. Creating a TDEKEK

The TDEKEK must be protected under the same OCS or softcard as that used to create the `tdeCredential` below.

To create a TDEKEK, or wrapping key, for database encryption:

```
USE master
CREATE ASYMMETRIC KEY dbAsymWrappingKey FROM PROVIDER <Name of provider>
WITH PROVIDER_KEY_NAME='ekmAsymWrappingKey', CREATION_DISPOSITION =
CREATE_NEW, ALGORITHM = RSA_2048;
GO
```

Where *<Name of provider>* is the name that is used to refer to the SQLEKM provider.

The TDEKEK is the only key you must create in the `master` database.

To check the TDEKEK, in SQL Server Management Studio navigate to **Databases > System Databases > Master > Security > Asymmetric Keys**. If necessary, right-click and select **Refresh**.

4.7.2. Setting up the TDE login and credential

1. In SQL Server Management Studio, navigate to **Security > Credentials**.
2. Right-click **Credentials**, then select **New Credential**.
3. Set **Credential name** to **tdeCredential** (for example).
4. Set **Identity** to *<OCSname>*, where *<OCSname>* is the name of the OCS or softcard. This must be the same key protector as that used to protect the `ekmAsymWrappingKey` created above.
5. Set **Password** to *<passphrase>*, where *<passphrase>* matches the passphrase on the OCS or softcard.
6. Set **Use Encryption Provider** to *<Name of provider>*, where *<Name of provider>* is the name of the SQLEKM provider you are using. Click **OK**.
7. In SQL Server Management Studio, navigate to **Security > Logins**.
8. Right-click **Logins**, then select **New Login**.
9. Set **Login name** to **tdeLogin** (for example).
10. Ensure **Mapped to asymmetric key** is selected, then select `dbAsymWrappingKey` (the TDEKEK created previously) from the drop down list.
11. Ensure **Map to Credential** is selected, then select **tdeCredential** from the drop down list. Click **Add**, then click **OK**.
12. In SQL Server Management Studio, check that the **tdeCredential** exists by navigating to **Security > Credentials**. If necessary, right-click and select **Refresh**. You should see the credential name listed.
13. In SQL Server Management Studio, check that the **tdeLogin** exists by navigating to **Security > Logins**. If necessary, right-click and select **Refresh**. You should see the login name listed.

4.7.3. Creating the TDEDEK and switching on encryption

Only one TDEDEK per database can be used at a time.

To create the TDEDEK using the `dbAsymWrappingKey` (TDEKEK) created above for database encryption, and enable TDE on the database (`TestDatabase`):

1. In SQL Server Management Studio, navigate to **Databases > TestDatabase**.
2. Right-click `TestDatabase`, then select **Tasks > Manage Database Encryption...**
3. Set **Encryption Algorithm** to the AES 256 algorithm.
4. Ensure that **Use server asymmetric key** is selected, then select `dbAsymWrappingKey` from the drop down list.
5. Ensure **Set Database Encryption On** is selected, then click **OK**.

After successfully setting up the TDE encryption, the person performing the set up no longer needs to use the same OCS or softcard for their login credential as used for the `tdeCredential`.

4.7.4. Verifying by inspection that TDE has occurred on disk

Note that the inspection method will only work for data that can be backed up in the database (on disk) as human-readable character strings.

To check the encryption state of the database, refer to How to check the TDE encryption/decryption state of a database. If the TDE has been successful, then an 'Encrypted' state should be indicated.

Querying the `TestTable` or database contents will not indicate whether the table was encrypted on disk, because it will be automatically decrypted when loaded into memory. TDE encryption on disk can be verified by inspecting backup copies of the `TestDatabase` from before and after the TDE encryption.

After TDE encryption has been set up and checked to be functioning, make a backup copy of the encrypted `TestDatabase`: see Backing up a database with SQL Server Management studio for instructions.

You should now have the following unencrypted and encrypted backup copies of the `TestDatabase`:

- `<Drive>:\<Backup_directory_path>\TestDatabase_TDE_Unencrypted.bak`
- `<Drive>:\<Backup_directory_path>\TestDatabase_TDE_Encrypted.bak`

These backup files can be inspected using a simple text editor, provided you have appropriate access permissions.

1. Open `TestDatabase_TDE_Unencrypted.bak` in a text editor and search for a known value. It should be possible to find the plaintext `FirstName` or else `LastName` of anyone mentioned in the original and unencrypted `TestTable`.
2. Open `TestDatabase_TDE_Encrypted.bak` in a text editor and search for the same value. It should not be possible to find any plaintext names or other values in the encrypted file. The backup files circumvent the automatic TDE decryption of the database, allowing direct inspection of the contents as stored on disk. Although this inspection has been carried out on backup files, these should contain information similar enough to the actual database disk contents to demonstrate whether the TDE encryption is working on disk or not.

4.7.5. To replace the TDEKEK

1. Following the procedure above (see Creating a TDEKEK) create a new asymmetric TDEKEK called `dbAnotherAsymWrappingKey`.
2. Create the new credential `anotherTdeCredential`.
3. Create a new TDE login called `anotherTdeLogin`. Map it to `dbAnotherAsymWrappingKey` and the new `anotherTdeCredential`.
4. In SQL Server Management Studio, navigate to **Databases > TestDatabase**.
5. Right-click TestDatabase, then select **Tasks > Manage Database Encryption...**
6. Select **Re-Encrypt Database Encryption Key** and **Use server asymmetric**. Select `dbAnotherAsymWrappingKey` from the drop down list.
7. Ensure **Regenerate Database Encryption Key** is not selected.
8. Ensure **Set Database Encryption On** is selected, then click **OK**.

4.7.6. To replace the TDEDEK

1. In SQL Server Management Studio, navigate to **Databases > TestDatabase**.
2. Right-click **TestDatabase**, then select **Tasks > Manage Database Encryption...**
3. Ensure **Re-Encrypt Database Encryption Key** is not selected.
4. Ensure **Regenerate Database Encryption Key** is selected, then select **AES 256** from the drop down list.
5. Ensure **Set Database Encryption On** is selected, then click **OK**.

4.7.7. Switching off and removing TDE

See Uninstalling and upgrading.

4.7.8. How to check the TDE encryption/decryption state of a database

The following `encryption_state` information applies to TDE encryption only.

You can use the following T-SQL queries to find the current encryption state of a database. This can be particularly useful where large amounts of data have to be processed and you wish to check progress before attempting any further operations on the database.

First, find the database ID from the database name by using the following query:

```
SELECT DB_ID('<Database name>') AS [Database ID];
GO
```

Where *<Database name>* is the name of the database you are interested in.

List database encryption states by using the following query:

```
SELECT * FROM sys.dm_database_encryption_keys
```

The above query provides a table output that includes columns titled `database_id` and `encryption_state`.

Find the database ID you are interested in and look at the corresponding value for the encryption state.

Alternatively, you can use the composite query:

```
SELECT db_name(database_id), encryption_state
FROM sys.dm_database_encryption_keys
```

Where database_id is the ID number of the database you are interested in.

Values of `encryption_state` are as follows:

Value of `encryption_state`	Meaning of value
0	Encryption disabled (or no encryption key)
1	Unencrypted or Decrypted
2	Encryption in progress
3	Encrypted
4	Key change in progress
5	Decryption in progress
6	Protection change in progress (The certificate or asymmetric key that is encrypting the database encryption key is being changed.)

4.8. Cell Level Encryption (CLE)

In CLE separate data fields in the same table can be encrypted under different encryption keys. These keys can be protected by different credentials. Unlike TDE protection, the user will need to obtain keys from the SQLEKM provider, and must have the correct credential to

authorize and load the encryption key(s) for the specific encrypted data they wish to access. Non-encrypted data is not affected by this and is visible to any authorized user.

Cell-level encryption will only work on data stored in the database as VARBINARY type. You must provide any necessary type conversions so that data is in VARBINARY form before encryption is performed. Decryption will return the data to its original VARBINARY structure. It may then be necessary to reconvert to its original type for viewing in human-readable form.

Database backup files that use the VARBINARY type are not human-readable. Therefore, the previous inspection method, as used for TDE to directly check if data has been encrypted on disk, cannot be used for cell-level encryption.

If you have not already created the following keys and made them available in your current database copy, then create them now.

4.8.1. Symmetric key

```
USE TestDatabase
CREATE SYMMETRIC KEY dbAES256Key
FROM PROVIDER SQLEKM
WITH PROVIDER_KEY_NAME='ekmAES256Key',
IDENTITY_VALUE='Rg7n*9mnf29xl4',
CREATION_DISPOSITION = CREATE_NEW, ALGORITHM=AES_256;
GO
```

4.8.2. Asymmetric key

```
USE TestDatabase
CREATE ASYMMETRIC KEY dbRSA2048Key FROM PROVIDER SQLEKM
WITH PROVIDER_KEY_NAME='ekmRSA2048Key',
CREATION_DISPOSITION = CREATE_NEW, ALGORITHM=RSA_2048;
GO
```

4.8.3. Encrypting and decrypting a single cell of data

Before you start, make sure you have a fresh version of the `TestTable` that is unencrypted.

In the example below, the encrypted and decrypted data is stored separately. Normally, the original data would be overwritten with the processed data.

1. View `TestTable` by running the following query:

```
SELECT TOP 10 [FirstName]
,[LastName]
,CAST(NationalIdNumber AS decimal(16,0)) AS [NationalIDNumber]
,(NationalIdNumber) AS VarBinNationalIdNumber
,[EncryptedNationalIdNumber]
,[DecryptedNationalIdNumber]
FROM [TestDatabase].[dbo].[TestTable]
```

   You will see the column `NationalIdNumber` in its original decimal form, and the column `VarBinNationalIdNumber` which shows the same number in its VARBINARY form (as stored in the database), and in which it will be encrypted.

   The columns `EncryptedNationalIdNumber` and `DecryptedNationalIdNumber` should contain NULL.

2. To encrypt a single cell in the `TestTable`, run the following query:

```
USE TestDatabase
UPDATE TestTable
SET EncryptedNationalIDNumber = EncryptByKey(Key_GUID('dbAES256Key'),
NationalIDNumber)
WHERE FirstName = 'Kate' AND LastName = 'Austin';
GO
```

   This query encrypts the `NationalIdNumber` for Kate Austin using the symmetric encryption key `dbAES256Key`, and stores the result in the column `EncryptedNationalIDNumber`.

   If encrypting using an asymmetric key, run the following query:

```
USE TestDatabase
UPDATE TestTable
SET EncryptedNationalIDNumber =
ENCRYPTBYASYMKEY(ASYMKEY_ID('dbRSA2048Key'), NationalIDNumber)
WHERE FirstName = 'Kate' AND LastName = 'Austin';
GO
```

3. Run the previous `View Table` query. The `EncryptedNationalIdNumber` will now contain the encrypted value against the name Kate Austin.
4. Run the following query to decrypt the information:

```
USE TestDatabase
UPDATE TestTable
SET DecryptedNationalIDNumber = DecryptByKey(EncryptedNationalIDNumber)
WHERE FirstName = 'Kate' AND LastName = 'Austin';
GO
```

If decrypting using an asymmetric key, run the following query:

```
USE TestDatabase
UPDATE TestTable
SET DecryptedNationalIDNumber =
DECRYPTBYASYMKEY(ASYMKEY_ID('dbRSA2048Key'), EncryptedNationalIDNumber)
WHERE FirstName = 'Kate' AND LastName = 'Austin';
GO
```

5. Run the previous `View Table` query. The `DecryptedNationalIdNumber` will now contain the decrypted value against the name Kate Austin.

   Ensure that this value matches the corresponding value in the `VarBinNationalIdNumber` column. If the values match, then the decryption worked successfully.

6. To view the decrypted value in its original decimal form, run the following query:

```
SELECT TOP 10 [FirstName]
,[LastName]

,CAST(NationalIdNumber AS decimal(16,0)) AS [NationalIDNumber]
,(NationalIdNumber) AS VarBinNationalIdNumber
,[EncryptedNationalIdNumber]
,CAST(DecryptedNationalIdNumber AS decimal(16,0)) AS
[DecryptedNationalIdNumber]
FROM [TestDatabase].[dbo].[TestTable]
```

7. Reset the `EncryptedNationalIdNumber` and `DecryptedNationalIdNumber` columns by running the following query:

```
USE TestDatabase
UPDATE TestTable
SET EncryptedNationalIDNumber = NULL, DecryptedNationalIDNumber = NULL;
GO
```

4.8.4. Encrypting and decrypting columns of data

Before you start, make sure you have a fresh version of the `TestTable` that is unencrypted.

In the example below, the encrypted and decrypted data is stored separately. Normally, the original data would be overwritten with the processed data.

Perform the same steps as shown in Encrypting and decrypting a single cell of data, but in this case where encryption or decryption occurs, replace with the following queries.

- Encrypt an existing column of data using the symmetric key:

```
USE TestDatabase
UPDATE TestTable
SET EncryptedNationalIDNumber = EncryptByKey(Key_GUID('dbAES256Key'),
NationalIDNumber);
GO
```

- Decrypt an existing column of data using the symmetric key:

```
USE TestDatabase
UPDATE TestTable
SET DecryptedNationalIDNumber = DecryptByKey(EncryptedNationalIDNumber);
GO
```

- Encrypt an existing column of data using the asymmetric key:

```
USE TestDatabase
UPDATE TestTable
SET EncryptedNationalIDNumber = ENCRYPTBYASYMKEY(ASYMKEY_ID('dbRSA2048Key'), NationalIDNumber);
GO
```

- Decrypt an exUSE TestDatabase

```
UPDATE TestTable
SET DecryptedNationalIDNumber = DECRYPTBYASYMKEY(ASYMKEY_ID('dbRSA2048Key'), EncryptedNationalIDNumber);
GO
```

4.8.5. Creating a new table and inserting cells of encrypted data

The following assumes you have available `TestDatabase` and the keys `dbAES256Key`, `dbRSA2048Key` as created previously.

4.8.5.1. Create a table with an encrypted field:

To create a new database table `Customers`, where individual cells of data held in the third column (`CardNumber`) will be encrypted, execute the following query:

```
USE TestDatabase
GO
CREATE TABLE Customers (FirstName varchar(MAX), SecondName varchar(MAX), CardNumber varbinary(MAX));
```

4.8.5.2. Insert encrypted data with the symmetric key:

The following query allows the user to enter the sensitive data (`CardNumber`) via the keyboard and then immediately encrypt using a symmetric key, sending the `CardNumber` directly into memory (and database) in an encrypted state.

```
USE TestDatabase
INSERT INTO Customers (FirstName, SecondName, CardNumber)
VALUES ('Joe', 'Bloggs', ENCRYPTBYKEY(KEY_GUID('dbAES256Key'),
CAST('<16 digit card number>' AS VARBINARY)));
INSERT INTO Customers (FirstName, SecondName, CardNumber)
VALUES ('Iain', 'Hood', ENCRYPTBYKEY(KEY_GUID('dbAES256Key'),
CAST('<16 digit card number>' AS VARBINARY)));
INSERT INTO Customers (FirstName, SecondName, CardNumber)
VALUES ('Joe', 'Smith', ENCRYPTBYKEY(KEY_GUID('dbAES256Key'),
CAST('<16 digit card number>' AS VARBINARY)));
GO
```

where *<16 digit card number>* is a 16-digit payment card number to be encrypted.

4.8.5.3. View data encrypted with the symmetric key in plain text:

The following query allows the user to view, in plain text on screen, the sensitive data (`CardNumber`) for customers named 'Joe'. The data remains encrypted in memory and (database).

```
USE TestDatabase
SELECT [FirstName], [SecondName],
CAST(DecryptByKey(CardNumber) AS varchar) AS 'Decrypted card number'
FROM Customers WHERE [FirstName] LIKE ('%Joe%');
GO
```

If an asymmetric key (`dbRSA2048Key`) is used, similar actions can be achieved using the following queries.

4.8.5.4. Insert encrypted data with the asymmetric key:

```
USE TestDatabase
INSERT INTO Customers (FirstName, SecondName, CardNumber)
VALUES ('Joe', 'Connor', ENCRYPTBYASYMKEY(ASYMKEY_ID('dbRSA2048Key'),
CAST('<16 digit card number>' AS VARBINARY)));
INSERT INTO Customers (FirstName, SecondName, CardNumber)
VALUES ('Richard', 'Taylor', ENCRYPTBYASYMKEY(ASYMKEY_ID('dbRSA2048Key'),
CAST('<16 digit card number>' AS VARBINARY)));
INSERT INTO Customers (FirstName, SecondName, CardNumber)
VALUES ('Joe', 'Croft', ENCRYPTBYASYMKEY(ASYMKEY_ID('dbRSA2048Key'),
CAST('<16 digit card number>' AS VARBINARY)));
GO
```

where *<16 digit card number>* is a 16-digit payment card number to be encrypted.

4.8.5.5. View data encrypted with the asymmetric key in plain text:

```
USE TestDatabase
SELECT [FirstName], [SecondName],
```

```
CAST(DECRYPTBYASYMKEY(ASYMKEY_ID('dbRSA2048Key'),CardNumber) AS varchar) AS 'Decrypted card number'
FROM Customers WHERE [FirstName] LIKE ('%Joe%');
GO
```

It is possible to encrypt separate table cells using different keys. When decrypting with a particular key, it should not be possible to see data that was encrypted using another key.

4.9. Viewing tables

4.9.1. Using SQL Server Management Studio

To check that data in a table was either encrypted or decrypted successfully, complete the following steps:

1. Open SQL Server Management Studio on the Management Studio.
2. Go to **Databases > TestDatabase > Tables**.
3. Right-click the table name and select **Select Top 1000 Rows** to view the encrypted or decrypted data.

4.9.1.1. Using SQL Query

To check that data in a table was either encrypted or decrypted successfully, execute the following SQL query:

```
Use TestDatabase
SELECT * FROM <table_name>
```

4.10. Checking keys

The following queries show how you can check the attributes of keys in your database and SQLEKM provider.

- To view the symmetric keys in a database:

```
Use TestDatabase
SELECT * FROM sys.symmetric_keys
```

- To view the asymmetric keys in a database:

```
Use TestDatabase
```

```
SELECT * FROM sys.asymmetric_keys
```

- To list all Security World keys associated with the current set of credentials:

```
DECLARE @ProviderId int;
SET @ProviderId = (SELECT TOP(1) provider_id
FROM sys.dm_cryptographic_provider_properties
WHERE friendly_name LIKE '<Friendly_name_of_provider>');
SELECT * FROM sys.dm_cryptographic_provider_keys(@ProviderId);
GO
```

  Where *<Friendly_name_of_provider>* can be found as shown in Checking the configuration for the cryptographic provider you are using.

- To correlate symmetric keys between the database and cryptographic provider:

```
DECLARE @ProviderId int;
SET @ProviderId = (SELECT TOP(1) provider_id FROM sys.dm_cryptographic_provider_properties
WHERE friendly_name LIKE '<Friendly_name_of_provider>');
SELECT * FROM sys.dm_cryptographic_provider_keys(@ProviderId)
FULL OUTER JOIN sys.symmetric_keys
ON sys.symmetric_keys.key_thumbprint = sys.dm_cryptographic_provider_keys.key_thumbprint
WHERE sys.dm_cryptographic_provider_keys.key_type = 'SYMMETRIC KEY'
GO
```

  where *<Friendly_name_of_provider>* can be found in Checking the configuration for the cryptographic provider you are using.

- To correlate asymmetric keys between the database and cryptographic provider:

```
DECLARE @ProviderId int;
SET @ProviderId = (SELECT TOP(1) provider_id FROM sys.dm_cryptographic_provider_properties
WHERE friendly_name LIKE '<Friendly_name_of_provider>');
SELECT * FROM sys.dm_cryptographic_provider_keys(@ProviderId)
FULL OUTER JOIN sys.asymmetric_keys
ON sys.asymmetric_keys.thumbprint = sys.dm_cryptographic_provider_keys.key_thumbprint
WHERE sys.dm_cryptographic_provider_keys.key_type = 'ASYMMETRIC KEY'
GO
```

  where *<Friendly_name_of_provider>* can be found in Checking the configuration for the cryptographic provider you are using.

- To correlate all keys (symmetric and asymmetric) between the database and cryptographic provider:

```
DECLARE @ProviderId int;
SET @ProviderId = (SELECT TOP(1) provider_id FROM sys.dm_cryptographic_provider_properties
WHERE friendly_name LIKE '<Friendly_name_of_provider>');
SELECT * FROM sys.dm_cryptographic_provider_keys(@ProviderId)
FULL OUTER JOIN sys.symmetric_keys
ON sys.symmetric_keys.key_thumbprint = sys.dm_cryptographic_provider_keys.key_thumbprint
FULL OUTER JOIN sys.asymmetric_keys
```

```
ON sys.asymmetric_keys.thumbprint = sys.dm_cryptographic_provider_keys.key_thumbprint
GO
```

where *<Friendly_name_of_provider>* can be found in Checking the configuration for the cryptographic provider you are using.

4.10.1. Cross-referencing keys between the cryptographic provider and Security World

The same key may exist under a different name in the cryptographic provider and database, but will not be recognizable at all by direct inspection of keys in the Security World (`%NFAST_KMDATA%\local`, or `%NFAST_KMLOCAL%`).

The example below allows you to cross-reference the same key between the cryptographic provider and Security World. The key can in turn be cross-referenced to the same key in the database, as shown in previous examples.

If you are running a failover cluster, you will need to run these procedures on the active server.

In a command window, run the utility:

```
nfkminfo –l
```

You should see something similar to the example below:

```
Keys protected by softcards:
key_simple_sqlekm-ef990fcd2115d2784d04fc8c963cdefa3a184264-3d4790baa10b3537d84ffe97d2bb03d9008517db
'ekmAES256Key'
key_simple_sqlekm-ef990fcd2115d2784d04fc8c963cdefa3a184264-8789a2543b2fe980b172ca4616b680d76b51bfb4
'ekmRSA2048Key'
key_simple_sqlekm-ef990fcd2115d2784d04fc8c963cdefa3a184264-c9a4569eb22b54b15d91a4feebf5d1799f3750ed
'ekmWrappingKey'
Taking the first key as an example, we can divide the text fields as follows:
```

Field	String	Notes
Prefix	`key_simple_sqlekm-`	The key's prefix, showing that it is of type 'simple'.

Field	String	Notes
Protector	`ef990fcd2115d2784d04fc8c963cdefa3a184264-`	The hash that uniquely identifies the OCS card or softcard that is currently protecting the key within the cryptographic provider. In the example above, the `nfkminfo -l` output states that the key is protected by a softcard.
Key hash	`3d4790baa10b3537d84ffe97d2bb03d9008517db`	A unique identifier for that key within the cryptographic provider.
Friendly name	`ekmAES256Key`	The name entered by the user when the key was generated.

4.10.1.1. Detailed information about individual keys in the Security World

You can obtain detailed information about individual keys in the Security World by using the utility `nfkminfo –k <APPNAME> <IDENT>`.

To obtain detailed information about individual keys in the Security World; on a client server, first run the utility as `nfkminfo –k`. This will provide a list of keys under the headings `AppName` and `Ident` similar to the example below:

```
>nfkminfo -kKey list - 6 keys
AppName simple Ident sqlekm-ef990fcd2115d2784d04fc8c963cdefa3a184264-
139f8de8a1017c095079208a4ad3e1480bd10170
AppName simple Ident sqlekm-ef990fcd2115d2784d04fc8c963cdefa3a184264-262cad87e7b6836f4eb571c35c433f95eebeb4cc
AppName simple Ident sqlekm-ef990fcd2115d2784d04fc8c963cdefa3a184264-
56e6e89ddccf3218827ca49c53b8bf9d10a0e5dc
AppName simple Ident sqlekm-ef990fcd2115d2784d04fc8c963cdefa3a184264-70ffd81cbb650eac2b148b415d6a903fa6a35e6c
AppName simple Ident sqlekm-ef990fcd2115d2784d04fc8c963cdefa3a184264-
824bdf1e2a4fa1b30bc39bd52aa322fbc47e253a
AppName simple Ident sqlekm-ef990fcd2115d2784d04fc8c963cdefa3a184264-bbccc80048164e13deef3868dd4dfbadaf67856a
```

The `ident` should match the same key as seen in the Security World (`%NFAST_KMDATA%\local`, or `%NFAST_KMLOCAL%`).

Use the `AppName` and `Ident` information to obtain information about a specific key as shown in the example below:

```
C:\>nfkminfo -k simple sqlekm-ef990fcd2115d2784d04fc8c963cdefa3a184264-
bbccc80048164e13deef3868dd4dfbadaf67856a
Key AppName simple Ident sqlekm-ef990fcd2115d2784d04fc8c963cdefa3a184264-
bbccc80048164e13deef3868dd4dfbadaf67856a
BlobKA length 488
BlobPubKA length 0
BlobRecoveryKA length 856
name "ekmTripleDES3Key"
```

```
hash bbccc80048164e13deef3868dd4dfbadaf67856a
recovery Enabled
protection PassPhrase
other flags !PublicKey !SEEAppKey !NVMemBlob +0x0
softcard ef990fcd2115d2784d04fc8c963cdefa3a184264
gentime 2020-04-23 16:09:49
SEE integrity key NONE
BlobKA
format 6 Token
other flags 0x0
hkm c7da546dc01a093f53b6ef29e1eaec3b8d4454f6
hkt ef990fcd2115d2784d04fc8c963cdefa3a184264
hkr none
BlobRecoveryKA
format 9 UserKey
other flags 0x0
hkm none
hkt none
hkr 31edffe57fcca13b16835b32d989d8e7bce43a4b
No extra entries
```

What is called the `key_thumbprint` when viewing key information through T-SQL queries, is the same as the `hash` when viewed using the `nfkminfo` utility.

4.11. Changes in the SQLEKM provider require SQL Server restart

If changes are made solely to the SQLEKM provider or associated Security World, there is no automatic mechanism to transmit these changes to the SQL Server. In this case, after such changes have been made, the SQL Server must be restarted in order to recognize them.

Examples of changes within the SQLEKM provider or Security World that will necessitate a SQL Server restart are:

- Key import.
- OCS or softcard deletion.
- Passphrase changes.
- Insertion or removal of OCS cards from card reader (except where card presence is normally required for ongoing key authorization).
- Configuration changes affecting the Security World.

Where keys are created or deleted through SQL Server queries, a restart should not normally be required. You will require administrator rights to restart the SQL Server.

To restart the SQL Server:

1. In the SQL Server Management Studio, right-click on the server name and select **Restart**

   Or:

2. On a command line, enter the following commands in succession:

```
net stop mssqlserver
net start mssqlserver
```

System environment changes that affect SQL Server may also require a restart in order to be recognized.

5. Security World data and back-up and restore

Always make sure you have an up to date back-up of your Security World data that includes all files in the `local` folder. For information on backup and recovery of Security World data, see the *User Guide* for your HSM.

5.1. Disaster recovery

It should be part of your corporate disaster recovery policy to perform regular back-ups of both your database and associated Security World such that the back-ups remain up to date and synchronized with each other. For further information about backing up the Security World, see Backing up.

The back-up strategies you employ and how you implement them will depend on your particular corporate policies and requirements, and the specifics of the type of configuration you are using. This guide cannot cover all the potential options and complexities, and will only provide broad advice on backup and restoration using the supported forms of database encryption. Whichever back-up or restoration option you use, make sure you have safely tested it before putting it into practice.

When a Security World is created, an ACS cardset is created at the same time. You should choose a quorum of ACS cards in accordance with your corporate security policy. The total number of cards in the ACS cardset should include surplus cards in case of failure or loss of an ACS card. The ACS cards authorize loading of the Security World, and some management operations on its OCS cardsets and softcards (please see the *User Guide* for your HSM). You should always store your ACS cards in a secure location. Normally, you should not need to use the ACS cardset for everyday use with your SQLEKM provider. However, you may need to use it if you are restoring a Security World that was previously archived and must be reloaded onto an nShield HSM.

An OCS cardset is used to authorize use of encryption keys that are assigned to and protected by that OCS cardset. Softcards perform a similar function. There can be more than one OCS cardset and/or softcard. However, a softcard exists as a single entity and has only passphrase protection. Generally, an OCS cardset is considered more secure than a softcard because it can be created with a quorum of multiple cards, physical presence of the cards is required, and each card can be supplied with its own passphrase. However, these advantages may be somewhat constrained when used with a SQL Server credential, which entails a 1/N quorum and identical passphrase for every card in the OCS cardset for

the cards to be used interchangeably with the same credential.

The total number of cards in the OCS cardset should include surplus cards in case of failure or loss of an OCS card. Some of the cards should always be kept in a secure location, and access to OCS cards in everyday use should be restricted to authorized persons.

The presence of a protecting OCS card, or softcard, will be required when performing back-up or restoration operations for a TDE encrypted database. For cell encryption keys, the presence of a protecting OCS card or softcard should only be required for any preliminary encryption or decryption operations before back-up, but should not be required for back-up or restoration itself.

Encryption keys, OCS card data and softcard data, that are protected by the SQLEKM provider are stored in its Security World. Note, if using TDE encryption, this does not apply to the database encryption key (TDEDEK) which is stored as an integral part of the related database. However, it does apply to the TDE wrapping key (TDEKEK) which is used to protect the TDEDEK.

Note that the Security World will hold the encryption keys for ALL current databases it is being employed with. That may include encryption keys for databases you are not specifically backing up. Note also that it may hold encryption keys for the master database that are common to more than one user database. You may find it convenient that you need only one Security World back-up to cover several databases. Otherwise you will need to pursue a policy of one Security World for one database.

5.2. Backing up

Before backing up a database and corresponding Security World, make sure you are using versions of both that are synchronized to each other. That is, the Security World holds all the up to date and correct encryption keys that are being used by the matching database.

When performing back-ups, it is advised to back-up the database first, before backing up the Security World.

Take care you do not delete any encryption keys from the SQLEKM provider that you will later need for restoration. Check if you have keys with duplicate names in the SQLEKM provider. Although technically possible, permitting duplicate names in the SQLEKM provider is not advised as it leads to confusion and possible operational errors. To avoid any future problems with your back-up, if you have keys with duplicate names, consider methods to eliminate the duplicate names, such as re-encrypting data with differently named key(s), before back-up.

If you are backing up a database that uses cell encryption keys, you should ensure that all

sensitive data is encrypted first before back-up commences. Before back-up, remove the cell encryption key references from the database itself. If key references are not removed from the database, they will be stored within the database back-up. This should be avoided from a security point of view. If you are backing up a database that is both cell and TDE encrypted, perform the above instructions for the cell encryption keys before continuing with the following instructions for backing up a TDE encrypted database.

When backing up a TDE encrypted database, you must have the TDE credential (including OCS card or softcard) and database wrapping key (TDEKEK) present.

With TDE encryption, the database encryption key (TDEDEK) is an integral part of the related database. It is stored within the back-up, and not in the Security World. Note however, that the TDEDEK is protected by the TDEKEK which is held in the Security World.

If using a shared disk cluster, the exact same database and TDEDEK is being used irrespective of the currently active node. Hence it should not matter which node is currently active when a back-up is made. Similarly, if an availability group is being used with primary and secondary replicas (and no shared disk), the secondary replicas should use the same TDEDEK as the primary, and it should not matter which replica (or node) is being used during a back-up.

Once you have prepared the database as described above, you may back-up the database in a similar manner to an unencrypted database. If you are backing up a TDE encrypted database, it will be backed up while remaining in its encrypted form, which is advantageous from a security point of view. After you have backed up the database, you can then proceed to back-up the associated Security World folder.

The Security World data is encrypted and does not require any further encryption operation to protect it. It can only be used by someone who has access to a quorum of the correct ACS cards, OCS cards, softcards, their passphrases, an nShield HSM and nShield Security World Software. Therefore, back-up should simply consist of making a copy of the Security World file and placing the copy in a safe location.

You should not store back-up copies of the Security World in the same physical location as its corresponding database. You must keep a record of which database and which Security World back-ups correspond to each other, and where they are located.

You should also securely store and keep a record of ACS and OCS cards associated with each Security World, as necessary to restore the keys used by the database. If you are using many ACS or OCS cards, or many symmetric keys with an IDENTITY_VALUE attribute, you may consider securely documenting the associated passwords. Also, the more encryption keys in your Security World, the more necessary it becomes to record which keys are used to encrypt which data.

If you are backing up as part of a long term archive, and you are storing ACS and OCS cards for more than one Security World, make sure you have some way of clearly identifying which cards belong to which Security World.

Your backup will include data content of your selected database, but may not include backups of SQL Server logins or credentials. Please refer to Microsoft SQL Server documentation for details of how to back these up. Otherwise, when later restoring the database, you may have to recreate suitable SQL Server logins and credentials, although this should not be a difficult task.

5.2.1. Backing up a database with SQL Server Management studio

This provides a basic example of how to backup a database. Please refer to Microsoft SQL Server documentation for a more thorough treatment of backup (and restoration) of a database.

1. In SQL Server Management Studio, navigate to **Management**.
2. Right-click on **Management** and select **Back up**.
3. Set **Database_Name** using the pull down menu.
4. Set **Backup type** as **Full** using the pull down menu.
5. Set **Backup component** button as **Database**.
6. Under **Destination** select **Disk**.

Click **Remove** to set aside any previously named back-up file(s) that you do not want to keep. Click **Add** and provide a suitable path and name for the backup file, e.g. `<Drive>:<Backup_directory_path>\TestDatabase_TDE_[date].bak` (if you are using a database failover cluster, this path may be relative to the shared disk). Press **OK** to accept the file path and name. Press **OK** again. You must remove the existing entry as backup only allows a single entry to populate this field at any one time. Make sure that you rename with a meaningful and unique name for the Backup and include the `.bak` suffix.

7. When the back-up is complete, the message **The backup of database 'TestDatabase' completed successfully** is displayed. Press **OK**.
8. Make sure you can access the back-up file at the location given above.

If the database back-up fails with a message indicating that the transaction log is not up to date, repeat the above steps, but for

> step 4 select **Backup type** as **Transaction Log**. In step 6, provide a suitable **Log file name**. After this completes successfully, you should be able to perform the database back-up.

5.3. Restoring from a back-up

If you wish to restore from back-ups, make sure you are using corresponding database and Security World copies. Restore the Security World before restoring the corresponding database.

Essentially, restoring a Security World simply means restoring a back-up copy of the Security World folder. If the configuration has not changed, you need only restore the contents of the `local` folder. If the Security World you are restoring is not already loaded onto your HSM, you will then have to use its ACS cards and associated passphrases to load it.

Before restoring a Security World from a back-up, decide what you wish to do with any existing Security World that you may have in your `%NFAST_KMDATA%` or `%NFAST_KMLOCAL%` directory. If you wish to keep it, you may need to perform a back-up on it before proceeding.

If you are restoring a previous version of a Security World that still exists on your nShield HSM, then as a precaution in case of failure, make a local copy of the current Security World contents before proceeding. You may then either merge or replace the existing Security World with your back-up copy.

If you are restoring an archived Security World that no longer exists on your nShield HSM, you will need to use its ACS cards with passphrases in order to reload it. Refer to your *nShield HSM User Guide* for more information on loading an existing Security World.

Make sure that the Security World is restored on all nShield HSMs within your configuration. Once you have restored the Security World to the SQLEKM provider, **restart the SQL Server on the active or primary node you are using in order to pick up the changes**. After restoring the Security World, you can then go on to restore the corresponding database.

Restore the database, including a TDE encrypted database, in a similar manner to an unencrypted database.

Once the database is restored, you will require suitable SQL Server logins and associated credentials to use the database and retrieve keys from the Security World. If these are not already present, or you have not restored them by some independent means, you will need to regenerate them. In this case, to access the encryption keys you will need to create new credential(s) that incorporate the OCS cardset(s), or softcard(s), that protect the key(s)

you wish to use. Once you have created a credential you must associate it with an authorized login.

You can use the `rocs` facility to find out which keys in the Security World belong to which OCS cardset or softcard. You can then recreate SQL Server credentials accordingly. See the *User Guide* for your HSM for more details about the `rocs` utility. See Creating a credential for details of how to create a credential.

For cell encryption keys, once the database is restored with valid credentials and associated login, you can restore the cell encryption keys from the SQLEKM provider by reimporting them. But there is no need to do this until you need the keys. You must be using the correct credentials for the particular keys you wish to reimport, see Re-importing symmetric key or Re-importing an asymmetric key.

If you are restoring a database that uses both cell encryption and TDE encryption, then the database must first be restored for TDE encryption as shown below, before reimporting the cell encryption keys.

The following description focusses on restoring a TDE encrypted database. It assumes the database wrapping key (TDEKEK) has not been reimported into the master database.

Before proceeding to restore a TDE encrypted database:

- If you are attempting to restore a TDE encrypted database that is protected by an OCS based credential, insert the correct OCS card(s) into the nShield HSM card reader(s).
- The user will need to use a personal login that is associated through a credential with the same OCS or softcard that is protecting the TDEKEK for the database to be restored. If necessary, create a credential that uses this OCS or softcard, and associate it with the user login.

If using a shared disk cluster, you should only need to perform the following steps on the active node. If using an availability group (with no shared disk) you will need to perform the following steps on the primary and all secondary replicas.

- The database wrapping key (TDEKEK) should already exist in the Security World and you will need to reimport it into your master database using the 'OPEN_EXISTING' clause as in the example below.

```
USE master
CREATE ASYMMETRIC KEY dbAsymWrappingKey
FROM PROVIDER <Name of provider>
WITH PROVIDER_KEY_NAME='ekmAsymWrappingKey',
CREATION_DISPOSITION = OPEN_EXISTING;
GO
```

- You will need to recreate the TDE login and credential that was originally used with the database.

```
--OCS card example
USE master
CREATE LOGIN tdeLogin FROM ASYMMETRIC KEY dbAsymWrappingKey;
CREATE CREDENTIAL tdeCredential WITH IDENTITY = 'OCS1', SECRET = '+453X7V]MR'
FOR CRYPTOGRAPHIC PROVIDER SQLEKM;
ALTER LOGIN tdeLogin ADD CREDENTIAL tdeCredential;
GO
```

```
--Alternative Softcard example.
Use master
CREATE LOGIN tdeLogin FROM ASYMMETRIC KEY dbWrappingKey;
CREATE CREDENTIAL tdeCredential WITH IDENTITY = 'scard1', SECRET = '0O*dG0ffz2'
FOR CRYPTOGRAPHIC PROVIDER SQLEKM;
ALTER LOGIN tdeLogin ADD CREDENTIAL tdeCredential;
```

- If you are attempting to restore a TDE encrypted database that is protected by an OCS based credential, insert the correct OCS card(s) into the nShield HSM card reader(s).
- The user will need to use a personal login that is associated through a credential with the same OCS or softcard that is protecting the TDEKEK for the database to be restored. If necessary, create a credential that uses this OCS or softcard, and associate it with the user login.
- After setting up the TDEKEK and credentials above, you may now restore the TDE encrypted database in a similar manner to an unencrypted database. If the database was backed up in an encrypted state, it should be restored in an encrypted state, and you should not need to switch on encryption.

6. Troubleshooting

Problem / issue	Suggested diagnosis / solution
When you attempt to register the SQLEKM provider, an error message in Microsoft SQL Server Management Studio similar to the following is returned - **Msg 33029, Level 16, State 1, Line 1 Cannot initialize cryptographic provider. Provider error code: 1. (Failure - Consult EKM Provider for details)**	The Security World has become corrupted or unusable. You may not have correct permissions to use the Security World directory. If using a fail-over cluster with nShield Connects similar to the example shown, you will require both remote and shared directory permissions on the RFS host. If using a cluster with an RFS, make sure you have set the `%NFAST_KMLOCAL%` variable as a system variable, and not as a local variable.
Microsoft SQL Server Management Studio displays a message stating that a session could not be opened for the SQLEKM provider.	There is either no smart card in the card reader, or an incorrect smart card in the card reader. Alternatively, the wrong OCS name or passphrase has been entered into the credentials. If setting up or managing the TDE encryption keys, you must use the same OCS or softcard for your login credential as used for the `tdeCredential` to be created.
Microsoft SQL Server Management Studio displays a message stating that the key type property of the key returned by the SQLEKM provider does not match the expected value.	An attempt was made to create an asymmetric or a symmetric key with an unsupported algorithm.
After loss of communication with a remote HSM all database queries fail with an error.	Communications between the SQL Server and SQLEKM provider have failed to re-establish after loss. Restart the MS SQL Server. (You may need administrator privileges to do this.)
When viewing data in a table that is expected to be visibly encrypted or decrypted, the data is displayed as NULL.	You may be attempting to encrypt/decrypt data that requires a key you do not have permission to use under your current credential. You have not inserted an operator card, or you have the wrong operator card. You are attempting to view data in an unsuitable format.

Problem / issue	Suggested diagnosis / solution
You are using a **AlwaysOn** availability group and you see that a database is marked as (Not synchronizing/Recovery pending)	Possible causes are a permissions problem in accessing a database, or a secondary replica has not been successfully updated following changes to the primary. If you have recently altered your login credentials, check the credentials are correct, then restart the SQL Server instance that is not synchronized. If you think a replica has not updated correctly, try: • Running the script Resynchronizing in an availability group in T-SQL shortcuts and tips. • Update the database from the latest backup log.

7. Uninstalling and upgrading

If you delete a SQLEKM provider login credential you will no longer be able to use it for the SQLEKM provider.

If you delete an associated SQL Server login you will no longer be able to use it to access the SQL Server or SQLEKM provider and will be locked out.

7.1. Turning off TDE and removing TDE setup

You must turn off TDE on all your databases and remove TDE setup before uninstalling the nShield Database Security Option Pack. Otherwise, you will not be able to decrypt any databases encrypted with TDE.

Before disabling and removing TDE encryption you are advised to back up the encrypted database (see Backing up a database with SQL Server Management studio) and associated Security World.

1. In SQL Server Management Studio, navigate to **Databases > TestDatabase**.
2. Right-click **TestDatabase**, then select **Tasks > Manage Database Encryption...**
3. Ensure **Set Database Encryption On** is deselected, then click **OK**.
4. Wait for the decryption process to finish. Check this by referring to How to check the TDE encryption/decryption state of a database.
5. When the database has completed decryption, drop the encryption key using the following T-SQL query:

```
USE TestDatabase
DROP DATABASE ENCRYPTION KEY;
GO
```

6. Restart the database instance. If you are using a database failover cluster, you may have to do this directly on the active server. In SQL Server Management Studio right-click on the instance and select **Restart**.
7. In SQL Server Management Studio, navigate to **Security > Logins**, and select the TDE login you wish to remove (for example, **tdeLogin**). Right-click on the selected login and select **Properties**.
8. Ensure the associated credential (for example, **tdeCredential**) is highlighted then choose **Remove**. Untick the box **Map to credential**. Click **OK**.
9. In SQL Server Management Studio, navigate to **Security > Credentials**, and select the

same credential you previously removed from the login (for example, **tdeCredential**). Right-click on the credential and select **Delete**. In the following screen, select **OK**.

10. In SQL Server Management Studio, navigate to **Security > Logins**, and select the TDE login you wish to remove (for example, **tdeLogin**). Right-click on the selected login and select **Delete**. In the following screen, select **OK**.
11. In SQL Server Management Studio, navigate to **Databases > System Databases > master >Security > Asymmetric keys**.
    - Select the key you wish to remove (for example, **dbAsymWrappingKey**). Right-click on the key and select **Delete**.
    - Alternatively, you can use the following query:

```
USE master
DROP ASYMMETRIC KEY dbAsymWrappingKey REMOVE PROVIDER KEY;
GO
```

7.2. Uninstalling the nShield Database Security Option Pack

Do not uninstall the nShield Database Security Option Pack until you have:

- decrypted any data encrypted using the SQLEKM provider in all your databases
- turned off TDE.

To uninstall the nShield Database Security Option Pack:

1. Remove the loginCredential from the logged-in user:
    a. In SQL Server Management Studio, select **Security > Logins** and open up the properties of the logged-in user.
    b. Select **loginCredential**, then click **Remove**, then **OK**.
2. Select **Security > Credentials**, and delete the **loginCredential**.
3. Disable and remove the SQLEKM provider:
    a. Select **Security > Cryptographic Providers**.
    b. Right-click to select the SQLEKM provider and click **Disable Provider**.
    c. A dialog is displayed which shows that this action was successful. Click **Close**.
    d. Right-click to select the disabled SQLEKM provider, then click **Delete**, then **OK**.
4. Select **Start > Control Panel > Administrative Tools > Services** (or **Start > Administrative Tools > Services**, depending on your version of Windows). Select **SQL Server (MSQLSERVER)** and click **Action > Stop**.
5. Select **Start > Control Panel > Add/Remove programs** (or **Uninstall program**, depending on your version of Windows). Select **nShield Database Security Option**

Pack then click **Uninstall**.

6. A dialog is displayed asking if you want to continue with uninstalling the nShield Database Security Option Pack. Click **Yes**.
7. A setup status screen is displayed while the nShield Database Security Option Pack is uninstalled. Click **Finish** to complete the removal of the program from your system.
8. Select **Start > Control Panel > Administrative Tools > Services** (or **Start > Administrative Tools > Services**, depending on your version of Windows). Select **SQL Server (MSQLSERVER)** then click **Action > Start**.

7.3. Upgrading

Follow the instructions for your system:

- Upgrading a standalone system.
- Upgrading a clustered system.

7.3.1. Upgrading a standalone system

Enhancements will be made to the nShield Database Security Option Pack over time, and product upgrades made available to customers.

Upgrading is a service affecting operation.

Before upgrading, please confirm the following:

- The system and the database are in a working state.
- The version of the SQLEKM provider:

```
SELECT * FROM sys.dm_cryptographic_provider_properties;
```

To upgrade the product:

1. Disable the SQL provider using the following query:

```
ALTER CRYPTOGRAPHIC PROVIDER SQLEKM
DISABLE;
GO
```

2. Restart SQL Server.
3. Uninstall the existing nShield Database Security Option Pack:
    a. Ensure that you are signed in as **Administrator** or as a user with local administrator

rights.

b. Select **Start > Control Panel**.

c. Depending on your version of Windows, select **Add/Remove programs** or **Uninstall program**.

d. Select **nShield Database Security Option Pack**, then click **Uninstall**.

e. A dialog is displayed, asking if you want to continue with uninstalling the nShield Database Security Option Pack. Click **Yes**.

f. A setup status screen is displayed while the nShield Database Security Option Pack is uninstalled. Click **Finish** to complete the removal of the program from your system.

4. Install the upgraded version of the nShield Database Security Option Pack:

   a. Still signed in as **Administrator** or as a user with local administrator rights, use the provided installation media and launch `setup.msi` manually.

   b. Follow the onscreen instructions. Accept the license terms, and click **Next** to continue.

   c. The SQLEKM provider will be installed to `%NFAST_HOME%`. Click **Install** to start installation.

   d. Click **Finish** to complete the installation.

5. Retarget your existing keys using the `sqlekm_retarget_keys` tool.

   

   This will preserve your existing pkcs11 keys, creating copies usable by the new SQLEKM provider.

   Open a command prompt, and execute `sqlekm_retarget_keys`, specifying either `--all` (to retarget all existing pkcs11 keys) or `--key-idents <the list of key idents>` (to retarget only the specified pkcs11 keys). For example:

   ```
   sqlekm_retarget_keys --all
   ```

6. Specify the new SQLEKM provider with the following query (specifying the path as appropriate):

   ```
   ALTER CRYPTOGRAPHIC PROVIDER SQLEKM
   FROM FILE = 'C:\Program Files\nCipher\nfast\bin\ncsqlekm.dll';
   GO
   ```

7. Enable the new SQLEKM provider using the following query:

   ```
   ALTER CRYPTOGRAPHIC PROVIDER SQLEKM
   ENABLE;
   ```

```
GO
```

8. Restart SQL Server and check that the system and database are in a working state. Confirm that the version of the SQLEKM provider is as expected with the following query:

```
SELECT * FROM sys.dm_cryptographic_provider_properties;
```

7.3.2. Upgrading a clustered system

Enhancements will be made to the nShield Database Security Option Pack over time, and product upgrades made available to customers.

Upgrading is a service affecting operation.

Before upgrading, please confirm the following:

- The system and the database are in a working state.
- Each of the nodes can become the active node.
- The version of the SQLEKM provider:

```
SELECT * FROM sys.dm_cryptographic_provider_properties;
```

The procedure is based on upgrading a passive node. This means that the particular node will have to be taken out of service. The following steps are dependent on being able to pause a passive node in the cluster, using the Failover Cluster Manager via the Windows Server Manager (**Server Manager Dashboard > Tools > Failover Cluster Manager**). The paused node is the one to be upgraded. Windows Server 2016 and later versions are automatically cluster-aware. Therefore the Failover Cluster Manager should be available on all versions of Windows that can run SQL Server Enterprise.

To upgrade the product:

1. Select a passive node (Node U) that will be temporarily out of service while it is upgraded. At least one other node (Node A) must remain active to continue service.
2. Using the Failover Cluster Manager (**Server Manager Dashboard > Tools > Failover Cluster Manager**), open the Nodes tree, and select Node U. Pause Node U by selecting **Pause > Drain roles**.
3. On node U, disable the SQL provider using the following query:

```
ALTER CRYPTOGRAPHIC PROVIDER SQLEKM
DISABLE;
```

```
GO
```

4. On node U, uninstall the existing nShield Database Security Option Pack:
   a. Ensure that you are signed in as **Administrator** or as a user with local administrator rights.
   b. Select **Start > Control Panel**.
   c. Depending on your version of Windows, select **Add/Remove programs** or **Uninstall program**.
   d. Select **nShield Database Security Option Pack**, then click **Uninstall**.
   e. A dialog is displayed, asking if you want to continue with uninstalling the nShield Database Security Option Pack. Click **Yes**.
   f. A setup status screen is displayed while the nShield Database Security Option Pack is uninstalled. Click **Finish** to complete the removal of the program from your system.
5. On node U, install the upgraded version of the nShield Database Security Option Pack:
   a. Still signed in as **Administrator** or as a user with local administrator rights, use the provided installation media and launch `setup.msi` manually.
   b. Follow the onscreen instructions. Accept the license terms, and click **Next** to continue.
   c. The SQLEKM provider will be installed to `%NFAST_HOME%`. Click **Install** to start installation.
   d. Click **Finish** to complete the installation.
6. On node U, retarget your existing keys using the `sqlekm_retarget_keys` tool.

   

   This will preserve your existing pkcs11 keys, creating copies usable by the new SQLEKM provider.

   Open a command prompt on node U, and execute `sqlekm_retarget_keys`, specifying either `--all` (to retarget all existing pkcs11 keys) or `--key-idents <the list of key idents>` (to retarget only the specified pkcs11 keys). For example:
   sqlekm_retarget_keys.py --all

7. On Node U, specify the new SQLEKM provider with the following query (specifying the path as appropriate):

```
ALTER CRYPTOGRAPHIC PROVIDER SQLEKM
FROM FILE = 'C:\Program Files\nCipher\nfast\bin\ncsqlekm.dll';
GO
```

8. On Node U, enable the new SQLEKM provider using the following query:

```
ALTER CRYPTOGRAPHIC PROVIDER SQLEKM
ENABLE;
GO
```

9. Using the Failover Cluster Manager (**Server Manager Dashboard > Tools > Failover Cluster Manager**), open the **Nodes** tree, and select Node U. Resume Node U by selecting **Resume > Do not fail roles**.
10. On Node U, check that the system and database are in a working state. Confirm that the version of the SQLEKM provider is as expected with the following query:

```
SELECT * FROM sys.dm_cryptographic_provider_properties;
```

11. Check that all databases held in common by both Node A and Node U are synchronized. If so, Node U can now be used for active service in the cluster if required.
12. Repeat all steps above, except step 6, for all remaining nodes that you wish to upgrade.

    Step 6 converts all pkcs11 keys in the security world to *nCore simple* keys. This only needs to be done once. Therefore, step 6 does not need to be repeated after the first node in a cluster has been upgraded.

8. T-SQL shortcuts and tips

The following T-SQL queries provide assistance or alternative methods to perform some of the examples shown in this document.

8.1. Creating a database

To create a database called `TestDatabase`.

```
USE master
GO
CREATE DATABASE TestDatabase;
GO
```

8.2. Creating a table

To create the following example table called `TestTable` within a previously created `TestDatabase`.

```
USE TestDatabase
GO
CREATE TABLE TestTable (FirstName varchar(MAX), LastName varchar(MAX),
NationalIdNumber varbinary(MAX), EncryptedNationalIdNumber varbinary(MAX),
DecryptedNationalIdNumber varbinary(MAX));
GO
INSERT INTO TestTable (FirstName, LastName, NationalIdNumber) VALUES ('Jack', 'Shepard', 156587454525658);
INSERT INTO TestTable (FirstName, LastName, NationalIdNumber) VALUES ('John', 'Locke', 2365232154589565);
INSERT INTO TestTable (FirstName, LastName, NationalIdNumber) VALUES ('Kate', 'Austin', 332652021154256);
INSERT INTO TestTable (FirstName, LastName, NationalIdNumber) VALUES ('James', 'Ford', 465885875456985);
INSERT INTO TestTable (FirstName, LastName, NationalIdNumber) VALUES ('Ben', 'Linus', 5236566698545856);
INSERT INTO TestTable (FirstName, LastName, NationalIdNumber) VALUES ('Desmond', 'Hume', 6202366652125898);
INSERT INTO TestTable (FirstName, LastName, NationalIdNumber) VALUES ('Daniel', 'Faraday', 7202225698785652);
INSERT INTO TestTable (FirstName, LastName, NationalIdNumber) VALUES ('Sayid', 'Jarrah', 8365587412148741);
INSERT INTO TestTable (FirstName, LastName, NationalIdNumber) VALUES ('Richard', 'Alpert', 2365698652321459);
INSERT INTO TestTable (FirstName, LastName, NationalIdNumber) VALUES ('Jacob', 'Smith', 12545254587850);
GO
```

8.3. Viewing a table

To view the previously created `TestTable`:

```
SELECT TOP 10 [FirstName]
,[LastName]
,[NationalIDNumber]
,[EncryptedNationalIdNumber]
,[DecryptedNationalIdNumber]
FROM [TestDatabase].[dbo].[TestTable]
```

To view the previously created `TestTable` with the `NationalIDNumber` in the original decimal form:

```
SELECT TOP 10 [FirstName]
,[LastName]
,CAST(NationalIdNumber AS decimal(16,0)) AS [NationalIDNumber]
,[EncryptedNationalIdNumber]
,[DecryptedNationalIdNumber]
FROM [TestDatabase].[dbo].[TestTable]
```

To view the previously created `TestTable` with the `NationalIDNumber` in the original decimal form, and also show the `NationalIdNumber` in `VarBinary` form:

```
SELECT TOP 10 [FirstName]
,[LastName]
,CAST(NationalIdNumber AS decimal(16,0)) AS [NationalIDNumber]
,(NationalIdNumber) AS VarBinNationalIdNumber
,[EncryptedNationalIdNumber]
,[DecryptedNationalIdNumber]
FROM [TestDatabase].[dbo].[TestTable]
```

8.4. Making a database backup

To make a database backup:

```
USE TestDatabase;
GO
BACKUP DATABASE TestDatabase
TO DISK = '<Drive>:\<Backup_directory>\TestDatabase_SomeState.bak'
WITH NOFORMAT,
INIT,
NAME = TestDatabase_SomeState Backup',
SKIP,
NOREWIND,
NOUNLOAD,
STATS = 10
GO
```

Where: *<Drive>:\<Backup_directory>* is the path to the directory to store the backup. If you are using a database failover cluster this will be relative to the active server.

8.5. Adding a credential

The following query will add a credential to the database:

```
CREATE CREDENTIAL <loginCredential> WITH IDENTITY = '<Credential name>', SECRET = '<Credential
passphrase>' FOR CRYPTOGRAPHIC PROVIDER
<Name of SQLEKM provider>;
ALTER LOGIN "<Domain>\<Login name>" ADD CREDENTIAL <loginCredential>;
```

Where

- *<loginCredential>* is the name you wish to provide for the credential.
- *<Credential name>* is the name of the OCS or softcard you wish to use as a credential.
- *<Credential passphrase>* is the passphrase of the OCS or softcard you wish to use as a credential.
- *<Name of SQLEKM provider>* is the SQLEKM provider name you are using.
- *<Domain>* is the relevant login domain.
- *<Login name>* is the relevant login name (to the database host).

8.6. Removing a credential

To remove a credential from the database:

```
ALTER LOGIN "<Domain>\<Login name>"
DROP CREDENTIAL <loginCredential>;
```

See Adding a credential for terms used.

8.7. Creating a TDEDEK

To create a TDEDEK using `TestDatabase` and `dbAsymWrappingKey` as an example:

```
USE TestDatabase;
CREATE DATABASE ENCRYPTION KEY WITH ALGORITHM = AES_256
ENCRYPTION BY SERVER ASYMMETRIC KEY dbAsymWrappingKey;
GO
```

8.8. Removing a TDEDEK

To remove a TDEDEK using `TestDatabase` as an example:

```
USE TestDatabase
DROP DATABASE ENCRYPTION KEY;
```

8.9. Switching on TDE

To switch on TDE using `TestDatabase` as an example:

```
ALTER DATABASE TestDatabase SET ENCRYPTION ON;
```

8.10. Switching off TDE

To switch off TDE using `TestDatabase` as an example:

```
ALTER DATABASE TestDatabase SET ENCRYPTION OFF;
```

8.11. Dropping a SQLEKM Provider

To drop the services of an existing SQLEKM Provider:

```
DROP CRYPTOGRAPHIC PROVIDER <Name of SQLEKM provider>
```

Where

- *<Name of SQLEKM provider>* is the name of an already existing SQLEKM Provider.

8.12. Disabling SQLEKM Provision

To disable the EKM provision in a SQL Server installation. This will disable all EKM providers:

```
sp_configure 'show advanced options', 1; RECONFIGURE;
GO
sp_configure 'EKM provider enabled', 0; RECONFIGURE;
GO
```

8.13. Resynchronizing in an availability group

To resynchronize a database called 'SourceDatabase' in an availability group, try:

```
USE master;
GO
ALTER DATABASE [SourceDatabase] SET HADR RESUME
```

8.14. Checking encryption state

To check the encryption state of your databases:

```
SELECT DB_NAME(e.database_id) AS DatabaseName, e.database_id, e.encryption_state, CASE e.encryption_state
```

```
WHEN 0 THEN 'No database encryption key present, no encryption'
WHEN 1 THEN 'Unencrypted'
WHEN 2 THEN 'Encryption in progress'
WHEN 3 THEN 'Encrypted'
WHEN 4 THEN 'Key change in progress'
WHEN 5 THEN 'Decryption in progress'
END AS encryption_state_desc, c.name, e.percent_complete FROM sys.dm_database_encryption_keys AS e
LEFT JOIN master.sys.certificates AS c ON e.encryptor_thumbprint = c.thumbprint
```