THALES

www.thales-esecurity.com

SQL Server 2014 Certified

Thales e-Security

nShield Microsoft SQL Server

Integration Guide

Version: 2.2

Date: 04 July 2016

Contents

Chapter 1: Introduction

This guide applies to the Thales Database Security Option Pack for Microsoft SQL Server® that must be activated for use with Thales nShield hardware security modules (nShield HSMs). It provides data-at-rest encryption for sensitive information held by Microsoft SQL Server.

Thales are pleased to announce that the Thales Database Security Option Pack for Microsoft SQL Server® has been tested and certified to meet Microsoft Gold standard for SQL Server 2014.

The product works in combination with nShield HSMs, Thales Security World Software, and Enterprise Editions of Microsoft® SQL Server® 2008, SQL Server® 2008 R2, Microsoft® SQL Server® 2012 and Microsoft® SQL Server® 2014 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 Thales SQLEKM provider supports Transparent Data Encryption (TDE) and Cell-Level Encryption (CLE) (and both concurrently), and also supports multithreaded operations. The nShield HSM is certified to the level of FIPS 140-2 to deliver a high level of security assurance. Its functions include protection of sensitive encryption keys and support for offload of encryption and key management operations.

This guide

The guide provides:

- An overview of how the Microsoft SQL Server, Thales Database Security Option Pack, Thales Security World software, and nShield HSMs may work together in order to enhance security.
- Installation instructions
- Configuration options
- Examples and advice on how the product may be used
- Troubleshooting advice
- Uninstall, and upgrade instructions

You can find the installer and all the associated configuration files and executables for the Database Security Option Pack for SQL Server on the supplied installation media.

This guide cannot anticipate all situations in which it may be desired to use the Thales SQLEKM provider. Example configurations and T-SQL scripts shown in this guide have all been tested to work and are given in good faith. However, these examples should be used primarily to learn how to use the SQLEKM provider, or adapted to your own circumstances. Thales accepts no responsibility for loss of data incurred by use of examples or any errors in this guide. For your own reassurance, it is recommended you thoroughly check your own solutions in safe test conditions before committing them to the production environment. If you require additional help in setting up your system, please contact Thales support.

Product configurations

The integration between the HSM and the SQLEKM provider has been tested for the following combinations:

Windows Server operating system version	Microsoft SQL Server version (Enterprise Edition)	Security World Software version	nShield Solo support	nShield Connect support
2012 R2 64-bit	2014 SP1	12.00	Yes	Yes
2012 R2 64-bit	2012 SP2	12.00	Yes	Yes
2012 R2 64-bit	2012 SP3	12.00	Yes	Yes
2012 R2 64-bit	2012 SP3	12.10	Yes	Yes
2008 R2 SP1 64-bit	2008 R2 SP3	12.00	Yes	Yes
2012 64-bit	2012	11.62	Yes	Yes
2008 R2 SP1 64-bit	2008 R2	11.62	Yes	Yes
2008 R2 SP1 64-bit	2012	11.50	Yes	Yes
2008 R2 64-bit	2008 R2	11.50	—	Yes
2008 64-bit	2008 R2	11.50	—	Yes
2008 32-bit	2008 R2	11.50	—	—
2003 R2 32-bit	2008 R2	11.50	Yes	—
2003 64-bit	2008 R2	11.50	Yes	—
2003 32-bit	2008	11.50	Yes	—

Note: It is not always possible to update the list of tested configurations immediately after a new SQL Server version, Service Pack, cumulative update or fix is released by Microsoft. However, provided the SQL Server EKM API remains unchanged, past history indicates it is very likely the SQLEKM provider will work with new updates. If a configuration is not listed here, this does not necessarily imply the configuration has not been tested, or is not supported, or will not work.

If a configuration is not listed above and you require more explicit information about tested configurations, or are having configuration problems, please contact Thales support.

If in doubt, we always recommend you use the latest update packs from Microsoft.

You should always test your configuration in a safe environment before committing to a production environment.

Supported platforms and environments

The Database Security Option Pack (or the SQLEKM provider) for SQL Server is fully compatible with V11.40 or higher of the Security World Software and a range of Thales nShield HSMs.

The SQLEKM provider supports the following Thales nShield HSMs:

- nShield Solo 10+, 500, 6000, 500+, and 6000+
- nShield Connect 500, 1500, 6000, 500+, 1500+, and 6000+.

The latest SQL Server service packs and cumulative updates may change. You should always check you are using the latest versions available from Microsoft, and update as necessary.

The SQLEKM provider has been tested to support the Enterprise Editions of:

- Microsoft SQL Server 2008 (with Service Pack 1)
- Microsoft SQL Server 2008 R2 (with Service Pack 3)
- Microsoft SQL Server 2012 (with Service Pack 3)
- Microsoft SQL Server 2012 (with Service Pack 2)
- Microsoft SQL Server 2014 (with Service Pack 1).

These are supported on the following platforms:

- Windows Server 2003 Enterprise Edition (32-bit and 64-bit configurations)
- Windows Server 2008 Enterprise Edition (32-bit and 64-bit configurations)
- Windows Server 2008 R2 Enterprise Edition (64-bit configuration)
- Windows Server 2012 Standard (64-bit configuration)
- Windows Server 2012 R2 Standard (64-bit configuration).

Supported Thales nShield functionality

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

Functionality	Support	Functionality	Support
Soft cards	Yes	Key Management	Yes
Strict FIPS (FIPS 140-2 Level 3) support	Yes	Key Recovery	Yes
Module Only Key	No	1 of N Card Set	Yes (see note 1)
Key Generation	Yes	Key Import	Partial (see note 2)
Fail Over	Yes	Load Balancing	Yes

Note: [1] K of N Card Set where K>1, is technically supported, but is not recommended, see *Appendix D: Using an OCS quorum of K/N where K>1* on page 82.

Note: [2] Key import is supported for pkcs11 keys only. Please see *Importing keys* on page 36.

Requirements

This guide assumes that:

- Your chosen version of Microsoft SQL Server is already installed. Your installation must include the latest service pack updates and hotfixes available from Microsoft.
- You are familiar with the administration and configuration of Microsoft SQL Server. This includes database clustering, if you wish to use it.
- You are familar with the T-SQL language and can perform basic SQL tasks such as creating a database or tables, etc.
- All users who wish to install, set up, configure or use the Thales Database Security Option Pack for SQL Server have a SQL Server login and appropriate permissions.
- You are familiar with the installation and configuration of Thales nShield Security World software and HSMs.
- You are familiar with database security concepts and practices. This guide provides provides basic examples of how to set up and use the Thales SQLEKM provider, but is not a primer on in-depth database security issues.

Terminology

To make this guide more straightforward to read:

- Microsoft SQL Server 2008, Microsoft SQL Server 2008 R2, Microsoft SQL Server 2012 and Microsoft SQL Server 2014 Enterprise Editions are referred to as simply SQL Server.
- The Thales nShield Security World software is referred to as the Security World software. It is a collection of programs and utilities that are used to administer, operate and maintain the Security World.

- The Security World means the HSM(s), ACS cards, OCS cards, softcards, encryption keys or other cryptographic material, that function in accordance to the Security World type. Often, when we talk about the Security World, we also imply the Security World software needed to make it function.
- Cryptographic files which represent ACS cards, OCS cards, softcards, encryption keys or other cryptographic material used by the Security World, are called Security World data, and are held in the Security World folder, see *Chapter 5: Security World Data and back-up/restore* on page 54.
- The Security World type refers to the characteristics of the chosen Security World. In this document only FIPS Level 2, or FIPS Level 3 will be mentioned. Please see your *HSM User Guide* for more information about Security World characteristics.
- The Database Security Option Pack for SQL Server working in conjunction with the Security World and Security World software is referred to as the SQLEKM provider (the Option Pack cannot function without the Security World).

Note: EKM is the Extensible Key Management (EKM) API provided for Microsoft SQL Server.

Where SQL Server Management Studio is referred to in the following text, any actions to be performed through its interface will normally be through its **SQL Server Management Studio** pane.

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. However, for convenience as a figure of speech, we may describe the keys that are referenced by the database as if they were loaded into the database or copied to the database.

With respect to SQL Server database clustering:

- If shared network drives are used, the active server is the cluster server currently in ownership of the shared drive.
- If an availability group is used with no shared drive, the active server is the one acting as the primary replica.

Commonly-used acronyms

ACS	Administrator Card Set
API	Application Programming Interface
CLE	Cell-Level Encryption
DLL	Dynamic Link Library
EKM	Extensible Key Management
FIPS	Federal Information Processing Standard (U.S.)
GUID	Global Unique Identifier
HSM	Hardware Security Module
ID	Identity
OCS	Operator Card Set
RFS	Remote File System
SQL	Structured Query Language
TDE	Transparent Data Encryption
TDEDEK	Transparent Data Encryption Database Encryption Key
TDEKEK	Transparent Data Encryption Key Encryption Key
T-SQL	Transact Structured Query Language

More information

For more information about:

- Installing a Thales nShield HSM, see the *Installation Guide* for your HSM
- Security World Software, see the appropriate *User Guide* for your HSM
- Microsoft SQL Server, visit the dedicated Microsoft web site at http://www.microsoft.com/sqlserver/
- Thales e-security as a Microsoft partner, see https://pinpoint.microsoft.com/en-us/companies/4295545937.

This guide forms one part of the information and support provided by Thales. You can find additional documentation in the `document` directory of the installation media for your product.

Contacting Support

To obtain support for your product, visit http://www.thales-esecurity.com/en/Support.aspx.

Before contacting the Support team, click **Useful Information** and use the subtopics to see the information that the team requires.

Chapter 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 Thales Database Security Option Pack for SQL Server supports this by including the security benefits of a nShield HSM and associated Thales Security World software. A brief description of how to perform encryption operations on Microsoft SQL Server using the SQLEKM provider is also given.

Note: 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.

Note: 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 1 provides a graphical overview of the cryptographic architecture outlined here.

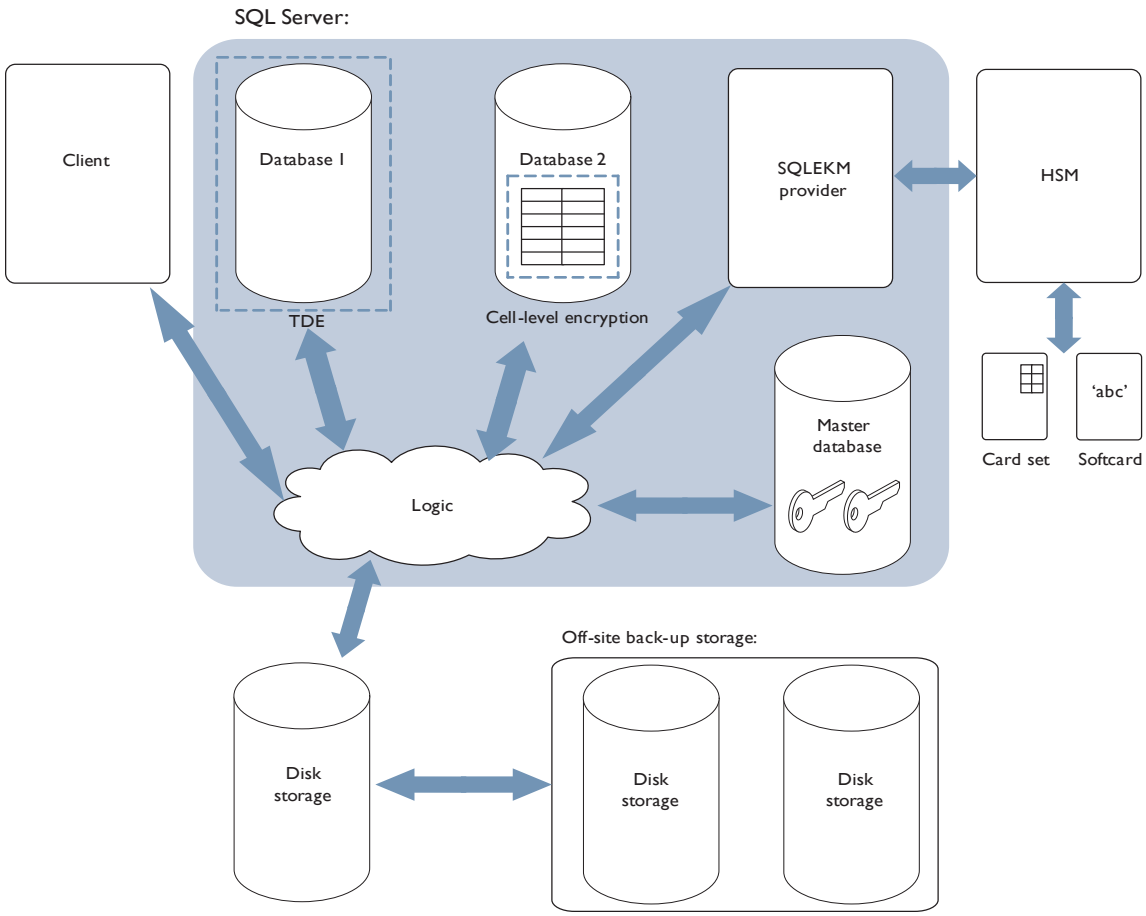

Figure 1. Cryptographic architecture

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

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.

Other benefits of using the Thales SQLEKM provider include:

- Ability to store keys from all 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) and Thales SQLEKM provider 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 Thales Security World holding the encryption keys.
- For CLE, knowledge of the encryption keys in use, and their passwords (if any).
- A Thales 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
- Electronic access to the encrypted database.

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.

Note: 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 organisation's security policies.

For more information about:

- Configuring the SQLEKM provider to perform encryption operations on SQL Server, see *Chapter 4: Configuring and using the SQLEKM provider* on page 25
- Restoration of Security World data from backup, see *Disaster recovery* on page 56.

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.

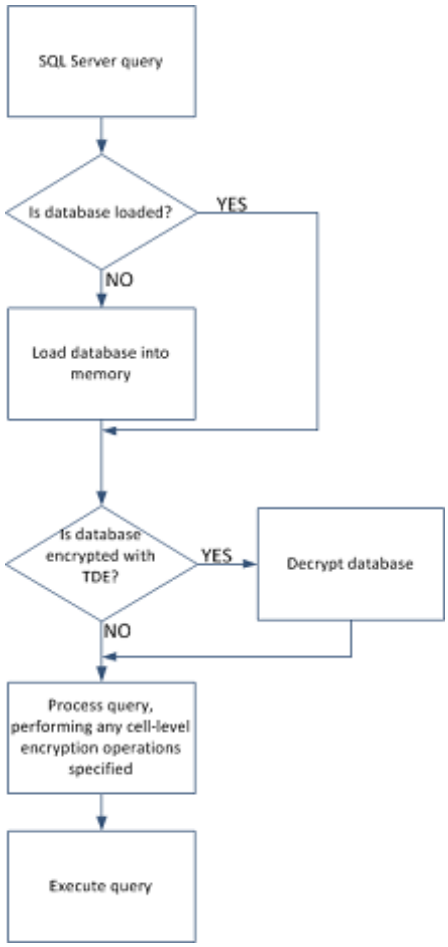

Figure 2. Querying encrypted data: process diagram

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.

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.

Example 1: TDE encryption/decryption only

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

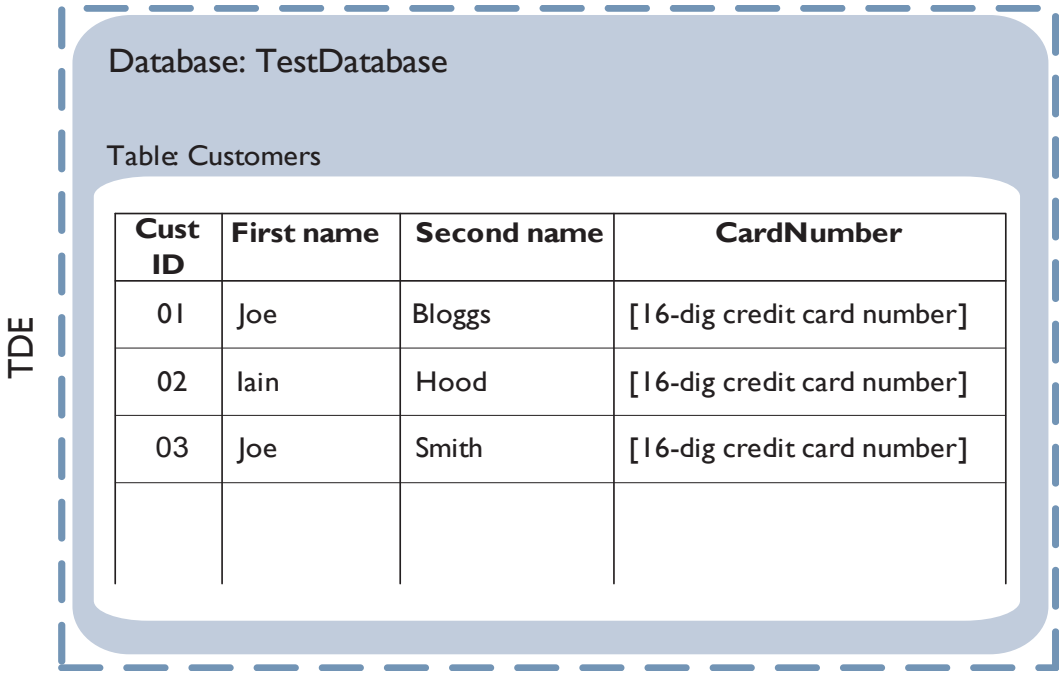

Figure 3. TDE encryption/decryption only

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%');
```

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.

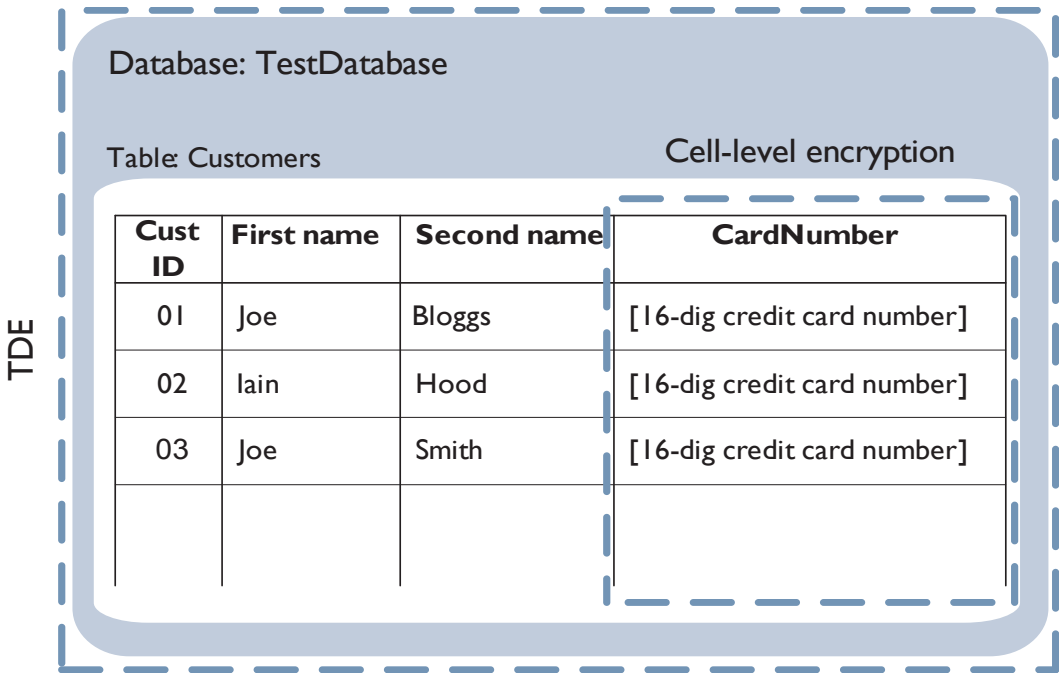

Figure 4. TDE and CLE/decryption

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%');
```

Chapter 3: System installation and configuration

This chapter describes:

- Installation and enabling the Database Security Option Pack for SQL Server to create the SQLEKM provider, including failover cluster examples.
- Configuring the SQLEKM provider for use.
- Generation of encryption keys.
- Use of both TDE and CLE on SQL Server databases.
- Encryption key checking or tracking.

For the SQLEKM provider software to function, it must be used in combination with Microsoft SQL Server software, a nShield HSM and Security World software. After these have been installed it also requires a usable Security World.

Supported platforms and environments

Refer to *Product configurations* on page 7 for information about supported platforms and environments.

Installation

The installation described here assumes the Microsoft SQL Server software is already installed. Ensure that all the latest service packs, updates and hotfixes for this 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.

During the installation process you may be required to create or reference environment variables used by the Security World software (e.g. **`NFAST_KMDATA`** or **`NFAST_KMLOCAL`**). For further information about environment variables, refer to the *User Guide* for your HSM.

The installer and associated configuration and executable files for the SQLEKM provider are on the supplied *Database Security Option Pack for SQL Server* installation disk.

To install the SQLKM provider as a stand-alone service, please refer to the section *Setting up as stand alone service* on page 18. To install the SQLKM provider within a database cluster environment, please refer to the section *Usage with database failover clusters* on page 18.

Setting up as stand alone service

Note: Before installing the Database Security Option Pack, you must install the Security World software. See the *User Guide* for your HSM for installation instructions.

To install the Database Security Option Pack for SQL Server:

1. Add `%NFAST_HOME%\toolkits\pkcs11` to the **`PATH`** environment variable.
2. Insert the installation disk DVD in your server drive. If it does not run automatically, launch **`setup.exe`** manually.
3. The **Welcome** screen of the InstallShield wizard is displayed. Click **Next**.
4. To accept the license agreement, click **Yes**. You also have the option to print the license agreement.
5. The SQLEKM provider software will be automatically installed to the default destination directory of **`%NFAST_HOME%`**.
6. A setup status screen is displayed, showing the progress of the installation. When the setup files finish installing, you are asked if you want to restart the machine now or later. To restart the machine at this point, select **Yes, I want to restart my computer now** and click **Finish**.
7. Add the following line to the `%NFAST_HOME%\cknfastrc` file:

```
CKNFAST_LOADSHARING=1
```

8. If you are intending to use DES or RSA_512 keys you should also add the following line to the `%NFAST_HOME%\cknfastrc` file:

```
CKNFAST_OVERRIDE_SECURITY_ASSURANCES=all
```

DES and RSA_512 keys are not recommended for use with Thales nShield products, and are not supported in some Security World types. For further information see *Supported cryptographic algorithms* on page 30.

9. If you are using an nShield Connect, configure the system as described in the *nShield Connect User Guide*.
10. If you do not have a Security World created, or loaded, on your server's HSM, you must do it now. You will also need an OCS cardset or softcard.
    See *Security Worlds, key protection and failover recovery* on page 23 if you are configuring a system to take into account automatic failure recovery.
11. See the *User Guide* for your HSM for instructions on checking the installation of the nShield PKCS #11 library with the **`ckcheckinst`** command line utility.

Usage with database failover clusters

The Thales SQLEKM provider can function as part of a Microsoft SQL Server database failover cluster. Two typical configurations are shown as examples that each incorporate a two-node failover cluster using a **shared disk**. A further example of configuring an *AlwaysOn* availability group with **no shared disk** for TDE encryption is given in *Appendix C: Using TDE within an AlwaysOn availability group* on page 73.

If you require assistance for different clustering arrangements, please contact Thales support.

If you are using shared disk arrangements, the first example in section *SQL Server database failover cluster using nShield Solo* on page 19, shows a configuration based on nShield Solo HSMs. The second example in section *SQL Server database failover cluster using nShield Connects* on page 21 shows a configuration that employs network based nShield Connect HSMs.

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

- Using the examples, the SQLEKM provider will be installed separately on each server in the cluster.
- The same version of the SQLEKM provider must be installed on each server. The version can be found by inspecting the **sqlekm** entry in the `version.txt` file that is part of the installation suite.
- In the example configurations, if failure occurs on either server 1 or server 2, then all database functionality including the SQLEKM provider will be transferred to the remaining server. There may be a short loss of service while the failover process completes. See *Security Worlds, key protection and failover recovery* on page 23 for discussion of which Security World type or protection to use.

SQL Server database failover cluster using nShield Solo

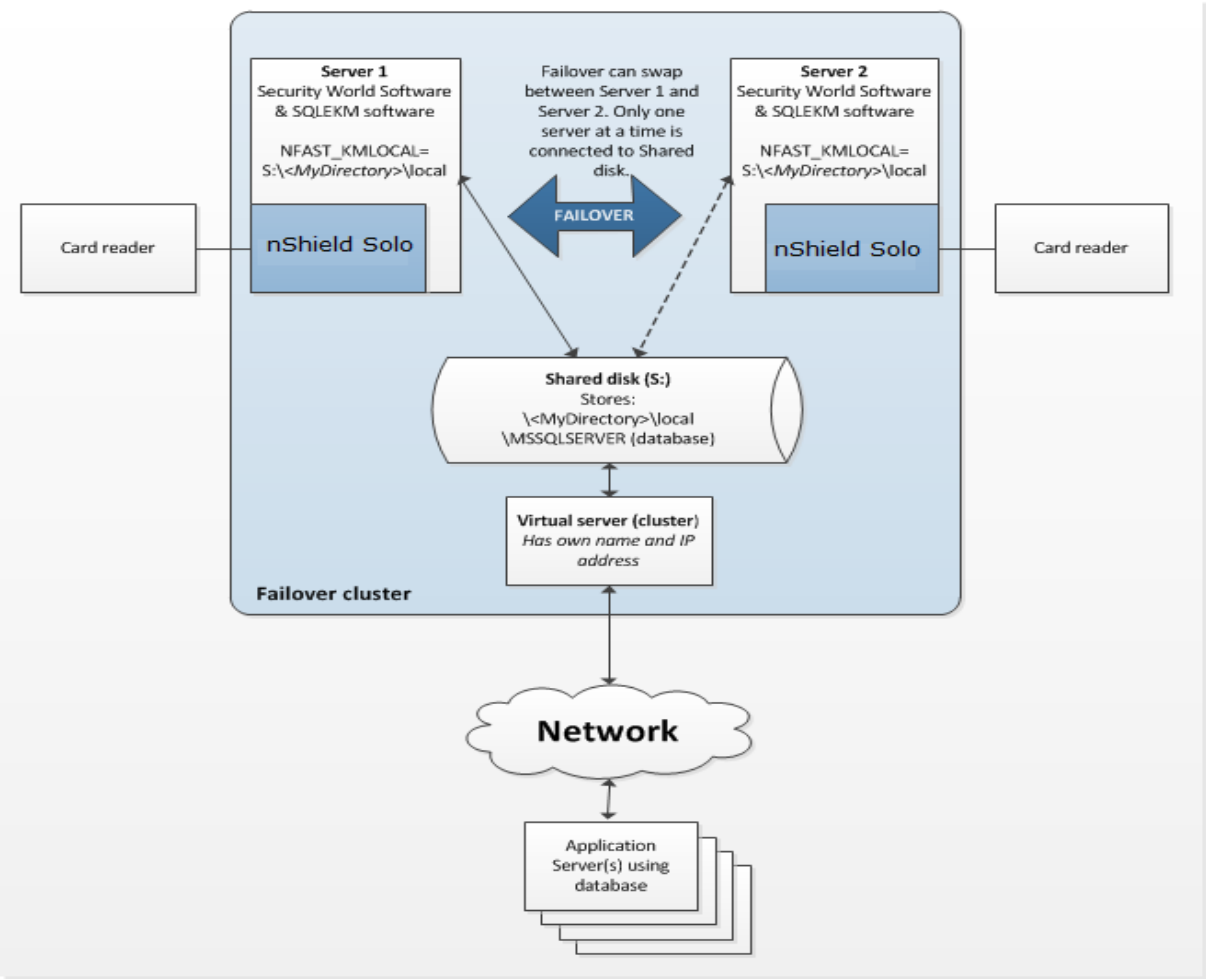

Figure 5. SQL Server database failover cluster using nShield Solo

Figure 5 shows a two node database failover cluster example that is configured to use nShield Solo based HSMs. To implement this configuration:

1. On server 1, complete the installation instructions in *Setting up as stand alone service* on page 18 (all steps, including Security World creation).
2. On server 2, complete steps 1 to 8 of the installation instructions in *Setting up as stand alone service* on page 18. Do *not* create a Security World on server 2.
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`
      Note: 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.
6. Use the **`nfkminfo`** utility to check the Security World on each server.
7. Before using the SQLEKM provider it must be enabled and a credential(s) set up as described later.

Note: If you have installed Thales V12.00 Security World software and you are using Java cards, be sure you have warranted your nShield Solo, and configured the cardlist file appropriately. In a cluster, you will need the same cardlist file contents on all servers in order to access the same cards. Please refer to the *User Guide* for your nShield HSM.

SQL Server database failover cluster using nShield Connects

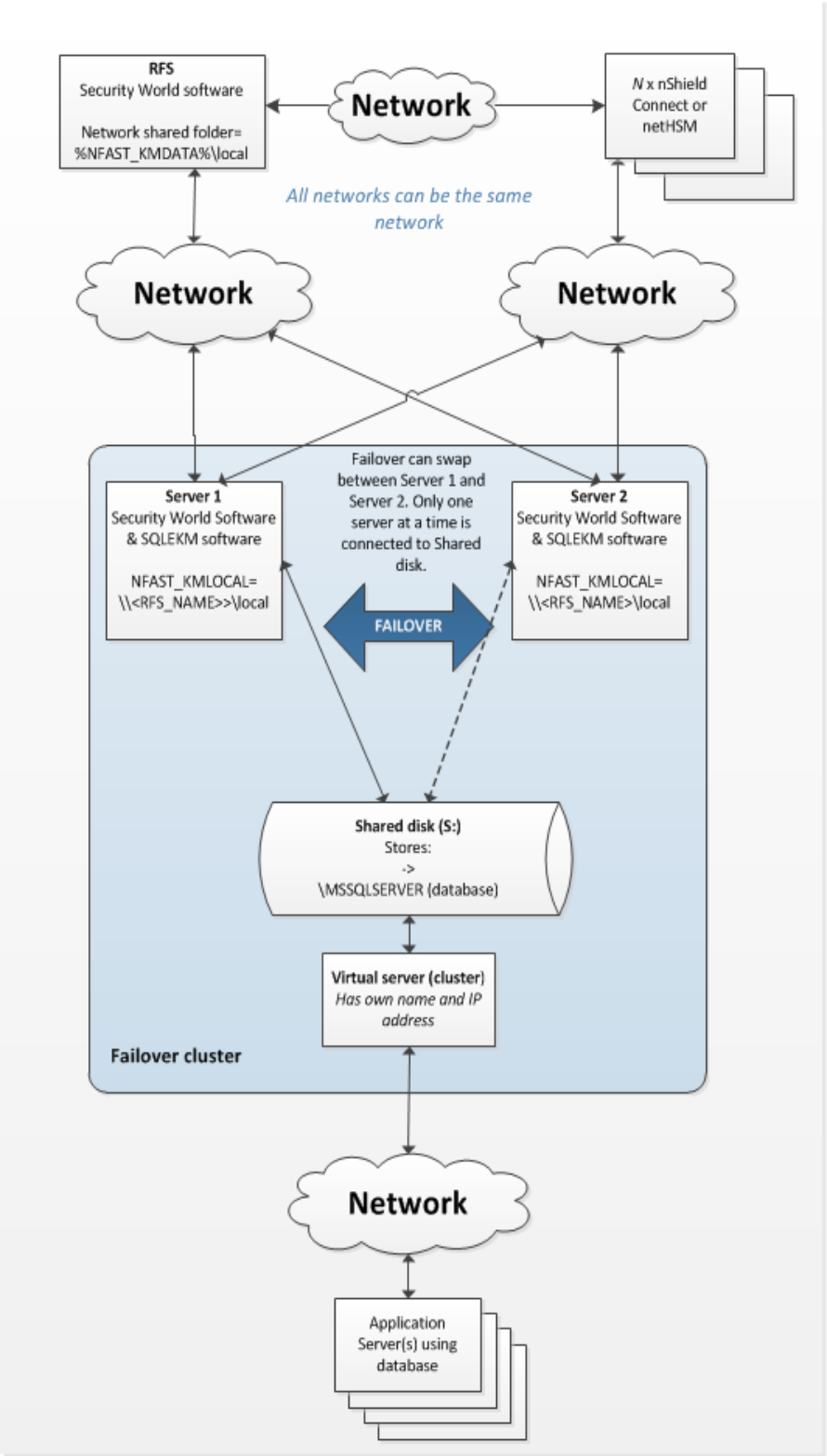

Figure 6. SQL Server database failover cluster using nShield Connects

Figure 6 shows a two node database failover cluster example using a shared disk that is configured to use nShield Connects. 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 *Appendix C: Using TDE within an AlwaysOn availability group* on page 73.

Note: In this example, if there is failure of the entire system (for instance a temporary power loss) then the RFS and nShield Connects 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 for any help.
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.
   Note: 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 steps 1 to 8 of the installation instructions in *Setting up as stand alone service* on page 18. Do *not* create a Security World on the servers.
4. On the 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`.
   Note: Make sure you *DO NOT* set this as a local variable.
   - 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.
8. Use the **`nfkminfo`** utility to check the Security World on each server and the RFS.
9. Before using the SQLEKM provider it must be enabled and a credential(s) set up as described later.

Note: If you have installed Thales V12.00 Security World software and you are using Java cards, be sure you have configured the cardlist file appropriately. In a cluster, you will need the same cardlist file contents on all servers in order to access the same cards. Please refer to the User Guide for your nShield HSM.

Security Worlds, key protection and failover recovery

This section briefly 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 World types and key protection please refer to the *User Guide* for your HSM.

Note: 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 strict FIPS [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 re-authorization after a failure, and permit automatic recovery. See *Creating a credential* on page 26 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.

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

Note: The softcards and OCS used must all be members of the same Security World.

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
Any	Module	Not supported	Not supported
FIPS Level 2	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.
FIPS 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.

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

Chapter 4: Configuring and using the SQLEKM provider

Note: In the example T-SQL statements featured in the remaining part of this guide, the names used for cryptographic keys (such as **dbAES256Key**) and databases (such as **TestDatabase**) are example names only. The only exception to this rule is the **master** database, which is a real database.

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.

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.

Please note:

- You must have an SQL Server login and appropriate permissions to configure or access the SQL Server or SQLEKM provider. You may need your system administrator to provide these.
- You must have a usable Security World loaded onto your server's HSM to register the SQLEKM provider. See *Installation* on page 17.

Note: If you have installed Thales V12.00 Security World software and you are using Java cards, then:

- be sure you have configured the cardlist file appropriately,
- if you are using an nShield Solo, be sure it is warranted.

Please refer to the User Guide for your nShield HSM.

Enabling the SQLEKM provider

To enable the SQLEKM provider on SQL server for both TDE and cell level encryption:

1. Ensure the following line exists in `%NFAST_HOME%\cknfastrc` :

```
CKNFAST_LOADSHARING=1
```

   Note: For a cluster configuration, this line must be present in the `cknfastrc` file on all servers (RFS and clients) within the cluster.

2. Enable support for SQLEKM providers within SQL Server by executing the following query:

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

3. Register the SQLEKM provider with the SQL Server 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.
- *<Path to provider>* is the fully qualified path to the `ncsqlekm.dll` file in the installation directory.

For example:

```
CREATE CRYPTOGRAPHIC PROVIDER SQLEKM
FROM FILE = 'C:\Program Files (x86)\nCipher\nfast\bin\ncsqlekm.dll';
GO
```

The SQLEKM provider installation wizard copies a 32-bit DLL into `ncsqlekm.dll` on 32-bit systems, and a 64-bit DLL into `ncsqlekm.dll` on 64-bit systems.
The alternative bit length version is named either `ncsqlekm32.dll` or `ncsqlekm64.dll`, depending on the installation platform.

4. 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>*, e.g SQLEKM.

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 SQLEKM provider. 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.

Note: 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.

Note: 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.

A SQL Server credential can support only one security token (OCS or softcard) at a time with one passphrase. The passphrase is stored within the credential and is required at set up of the credential only. If you are using an OCS cardset and wish to use the OCS cards interchangeably, they must all be programmed with the same passphrase and be from the same OCS cardset.

Note: We recommend that you always use a strong passphrase of at least 10 characters in length. However, you should also consult your organisation'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 and your organisation's security policies. 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 Thales `nfkminfo` utility as follows:

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

You will see the OCS card or softcard names (as exist) 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 *The local directory* on page 55 for more information about file hash numbers.

Note: 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* on page 37 and *Cell Level Encryption (CLE)* on page 42.

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>* e.g. SQLEKM. 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**.

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 (x86)\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* on page 30.

   Note: If a strict FIPS (FIPS 140-2 Level 3) Security World is used DES key type support is removed.

Encryption and encryption keys

When you have completed the configuration of 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 Thales 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.

Note: 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* on page 37.

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.

Note: 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.

Note: 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.

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 Thales 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 Thales **rocs** utility.

To use the examples in this document you will first need to create **TestDatabase** and **TestTable** as shown in *Creating a database* on page 67 and *Creating a table* on page 67. 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* on page 39 before adapting any examples. See also *Appendix A: T-SQL shortcuts and tips* on page 67.

Note: 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.

Note: You can check which keys are protected under which credential by using the Thales **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.

Note: 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.

Supported cryptographic algorithms

The algorithms that you can use for encryption depends on whether the Thales nShield HSM is compliant with the FIPS 140-2 Level 2 or the FIPS 140-2 Level 3 security standard.

For more information about cryptographic algorithms and FIPS 140-2 Level 3, see the *User Guide* for your HSM.

The following table lists cryptographic algorithms that you can use with symmetric keys.

Algorithm	FIPS 140-2 Level 2	FIPS 140-2 Level 3
DES	Yes	No
Triple_DES	Yes	Yes
Triple_DES_3KEY	Yes	Yes

Algorithm	FIPS 140-2 Level 2	FIPS 140-2 Level 3
AES_128	Yes	Yes
AES_192	Yes	Yes
AES_256	Yes	Yes

The following table lists cryptographic algorithms that you can use with asymmetric cryptographic keys.

Algorithm	FIPS 140-2 Level 2	FIPS 140-2 Level 3
RSA_512	Yes	Yes
RSA_1024	Yes	Yes
RSA_2048	Yes	Yes

Note: 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 Thales nShield products. You must modify the PKCS #11 library configuration file to use these keys. For more information, contact Thales Support.

Symmetric keys

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.

Note: The GUID issue does not apply to asymmetric keys.

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 *Appendix A: T-SQL shortcuts and tips* on page 67 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.

Note: 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*> should be a minimum of 10 characters in length and have qualities similar to a strong passphrase. Check your organisation'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.

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.

Creating and managing symmetric keys

Note: 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.

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 the section *Checking keys* on page 48.

Removing symmetric keys from the database only

To remove the symmetric key (**dbAES256Key**, created in the above procedure) 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.

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*9mnf29xl4',
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.

Removing symmetric keys from the database and provider

To remove a symmetric key (**dbAES256Key**) from both the database (**TestDatabase**) and the Thales 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.

Note: 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.

Creating and managing asymmetric keys

Note: The GUID issue that affects symmetric keys does not apply to asymmetric keys, and the *IDENTITY_VALUE* for GUID generation is not required.

Note: 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.

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.

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 the section *Checking keys on page 48*.

Removing an asymmetric key from the database only

To remove the asymmetric key (**dbRSA2048Key**, created in the above procedure) 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.

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
```

Removing an asymmetric key from the database and provider

To remove the asymmetric key (**dbAES256Key**) from both the database (**TestDatabase**) and the Thales 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.

Note: 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.

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.

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

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.

If a key was created by the SQLEKM provider independently of the SQL Server interface, you must restart SQL Server in order for the presence of the key to be registered, see *Changes in the SQLEKM provider require SQL Server restart* on page 53. Keys created in the SQLEKM provider can be imported into a database provided they are in pkcs11 format. Other formats will not be recognized by the database.

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. Thales 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 Thales support if you wish to pursue key import (or export) operations further.

The Security World permits pkcs11 key names with an arbitrary number of characters. However, if such a key is to be imported into an SQL Server database, the key name must be restricted to a maximum of 32 characters.

Note: The name length restriction here is slightly different from when creating the key through a T-SQL query, where the name length restriction is 31 characters maximum. See the section on *Symmetric keys* on page 31 or *Creating an asymmetric key* on page 34.

Note: After an externally created key has been placed in the Security World, you must restart the SQL Server before the key is imported for its presence to be recognized. See *Appendix A: T-SQL shortcuts and tips* on page 67

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 = CREATION_DISPOSITION = OPEN_EXISTING;
```

Where:

- *Your_database_name* is the name of the database for which you wish to provide encryption. See *Appendix A: T-SQL shortcuts and tips* on page 67 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.

Note: 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*> should be a minimum of 10 characters in length and have qualities similar to a strong passphrase. Check your organisation'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*>

Transparent Data Encryption - TDE

Note: An example of configuring an *AlwaysOn* availability group with no shared disk for TDE encryption is given in *Appendix C: Using TDE within an AlwaysOn availability group* on page 73.

These examples assume that both the **`TestDatabase`** and **`TestTable`** as described in *Appendix A: T-SQL shortcuts and tips* on page 67 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 (however, once the TDE encryption is working, they are free to remove the credential from their login).

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 TDE Key Encryption Key (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* on page 58. Alternatively, you may prefer to adapt the T-SQL query shown in *Making a database backup* on page 69. Save the backup as `<Drive>:\<Backup_directory_path>\TestDatabase_TDE_Unencrypted.bak`.

Note: 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 *Appendix C: Using TDE within an AlwaysOn availability group* on page 73.

Creating a TDEKEK

Note: 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**.

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 in the previous procedure) 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.

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.

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 the section *How to check the TDE encryption/decryption state of a database* on page 41. 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* on page 58 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.

To replace the TDEKEK

1. Following the procedure above (see *Creating a TDEKEK* on page 38) create a new asymmetric TDEKEK called **dbAnotherAsymWrappingKey**.
2. Create the new credential **anotherTdeCredential**.
3. Create a new TDE login called **anotherTdeLogin**. Map it to 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**.

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**.

Switching off and removing TDE

See *Uninstalling and Upgrading* on page 64.

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

Note: 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.)

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.

Even after the encryption keys are loaded into the database, a user who is authorized to use the database, but whose login is not associated with the correct credential, will not be able to use the keys.

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.

Note: 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.

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
```

Asymmetric key

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

Encrypting and decrypting a single cell of data

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

Note: 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:

   ### View Table:

   ```
   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:

   Encrypt a cell using the symmetric key:

```
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**.
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:

   Decrypt a cell using the symmetric key:

```
USE TestDatabase
UPDATE TestTable
SET DecryptedNationalIDNumber = DecryptByKey(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:

   View encrypted data:

```
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:

Reset table:

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

8. Repeat steps 1-7, using the asymmetric encryption key **dbRSA2048Key**.

Encrypt a cell using the asymmetric key

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

Decrypt a cell using the asymmetric key:

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

Encrypting and decrypting columns of data

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

Note: 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 the section *Encrypting and decrypting a single cell of data* on page 43, 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 existing column of data using the asymmetric key:

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

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.

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));
```

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.

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.

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.

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
```

Note: 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.

Viewing tables

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.

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>
```

Checking keys

The following queries show how you can check the attributes of keys in your database and SQLEKM provider. These methods are suitable for small numbers of keys. For large numbers of keys, seek automated methods.

- 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 view the keys in the 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);
GO
```

Where *<Friendly_name_of_provider>* can be found as shown in the section *Checking the configuration* on page 28 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 as shown in the section *Checking the configuration* on page 28 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 as shown in the section *Checking the configuration* on page 28 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 as shown in the section *Checking the configuration* on page 28 for the cryptographic provider you are using.

Cross-referencing keys between the SQLEKM provider and Security World

The same key may exist under a different name in the SQLEKM provider and database (see previous section), 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 SQLEKM provider and Security World. The key can in turn be cross-referenced to the same key in the database, as shown in previous examples.

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

1. In a command window, run the Thales utility:

```
cklist
```

   Note: You may have to enter the appropriate OCS or softcard passphrase.
2. In the command window, scroll through the keys displayed, and for each key observe its value of **CKA_LABEL**. This matches that key's name in the cryptographic provider, as specified by the user at key generation.

The **CKA_NFKM_ID** field has two parts:

- The prefix part of this is the identity of the protector (OCS or smartcard).
- The suffix part of this is the identity of the same key in the Security World.

Note: Under the **cklist** utility, each asymmetric key will appear as a separate public part and private part. The value of **CKA_NFKM_ID** should be identical for both parts.

Example:

In **cklist**

```
CKA_NFKM_ID =
"uc8930b1640cceca18dab54f6d304564d56a5263ebd9f590f9f9b40dd6d8effa29b640789f3a33f6a0"
```

Matches

```
key_pkcs11_
uc8930b1640cceca18dab54f6d304564d56a5263ebd9f590f9f9b40dd6d8effa29b640789f3a33f6a0
```

In the Security World.

Detailed information about individual keys in the Security World

You can obtain detailed information about individual keys in the Security World by using the Thales utility **nfkminfo -k <APPNAM> <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 -k
Key list - 6 keys
AppName pkcs11               Ident uc04fd373a8c273ff31fa2b715c82fafd62d9b0ebc-
4a937bcb08c4c10ddc10cb2e211e225f30076467
AppName pkcs11               Ident uc2d7bb2b4881ff0c2d6cdfb1d2d96495b836c99c3-
39782a17496647ac4bae4de4a2c73fc4114a0e11
AppName pkcs11               Ident uc2d7bb2b4881ff0c2d6cdfb1d2d96495b836c99c3-
e5c8491ea44f91a8bea1f4bdd71b2f7bd5a2bfd3
AppName pkcs11               Ident uc3ccec7b7a60fd737d3258561b62e8817a86bf0db-
20c01df1bff1e5233106be3be904885cb2f19754
AppName pkcs11               Ident uc5b92244580da11dd351a6c4538cc6515394eb8b2-
98049d55e508bc0014350aea7596d91926cf778c
AppName pkcs11               Ident uca6c59a0034c2c4762f70aaeeff0ce69ce620d863-
a05a30aa91506b51ffc8f1f52d52b521793a772f
```

The **ident** should match the same key as seen in the Security World (`%NFAST_KMDATA%\local`, or `%NFAST_KMLOCAL%`), or else the `CKA_NFKM_ID` as listed by the **cklist** utility.

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

```
>nfkminfo -k pkcs11 uc2d7bb2b4881ff0c2d6cdfb1d2d96495b836c99c3-
39782a17496647ac4bae4de4a2c73fc4114a0e11
Key AppName pkcs11 Ident uc2d7bb2b4881ff0c2d6cdfb1d2d96495b836c99c3-
39782a17496647ac4bae4de4a2c73fc4114a0e11
BlobKA length          1168
BlobPubKA length       516
BlobRecoveryKA length 1304
name                   "ekmWrappingKey"
hash                   0c7883d6b3cbd57ea3596f1efe2afe894317314e
recovery               Enabled
protection             CardSet
other flags            PublicKey !SEEAppKey !NVMemBlob +0x0
cardset                2d7bb2b4881ff0c2d6cdfb1d2d96495b836c99c3
gentime                2016-02-23 12:38:02
SEE integrity key      NONE


...etc...
```

Note: 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, or the **CKA_NFKM_HASH** when using the **cklist** utility.

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 an SQL Server restart are:

- Key creation or deletion
- Key import
- OCS creation or deletion
- Softcard creation or deletion
- Passphrase changes
- Insertion or removal of OCS cards from card reader (except where card presence is normally required for ongoing key authorisation)
- Addition or removal of modules
- 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
```

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

Chapter 5: Security World Data and back-up/restore

Operational data used by the Security World software is all the data held in the directory(s) referenced by the following environment variables:

- *%NFAST_KMDATA%*
- *%NFAST_KMLOCAL%* (typically only employed if a remote host is being used).

The *%NFAST_KMDATA%* variable will cover the following sub-directories under the `Key Management Data directory` (typically) on a local host:

- `config`
- `features`
- `hardserver.d`
- `local`
- `tmp`
- `warrants` (if using V12 software).

The *%NFAST_KMLOCAL%* variable will cover the `local` sub-directory under the `Key Management Data` directory (typically) on a remote host.

If you are using both the above variables at the same time, then the `local` sub-directory under the *%NFAST_KMDATA%* variable is superseded by the `local` sub-directory under the *%NFAST_KMLOCAL%* variable, which holds the relevant `local` data in this case.

The `local` sub-directory may also be called a Security World folder and holds the Security World data. This includes the cryptographic data files essential for the operation of the Security World. Cryptographic files in the `local` sub-directory may update or change regularly and cannot be replaced if lost. These files should be the focus of back-up.

The sub-directories other than `local` contain Security World configuration data. Once a configuration is established it is unlikely to change frequently. In any case, it is possible for the configuration data to be regenerated or replaced. Its loss may impede rapid restoration of a failed system, but the system should not be irrecoverable. Configuration files are not inherently encrypted. Information contained in them may give an adversary some knowledge of your configuration, but will not directly compromise the security of your cryptographic material. If you wish to keep configuration files secret you must do so using external encryption facilities.

Hence, a practical back-up strategy is to save an initial copy of the configuration sub-directories, and thereafter only update this back-up if the configuration is known to have changed. Regular or scheduled back-ups can then be confined to the `local` sub-directory contents.

All files that are held in the `local` folder are encrypted. If lost or stolen, they will be useless to anyone who does not possess the correct authorizing mechanisms to use them, such as ACS cards, OCS cards, HSM, associated passwords and Thales Security World software. Therefore back-up of the `local` data may simply consist of making a copy of it, and placing the copy in a safe location. No further encryption is necessary.

Further information about backing up the Security World can be found in the *User Guide* for your nShield HSM.

The local directory

The `local` sub-directory, or Security World folder, contains the files (or Security World data) needed to perform the cryptographic functions of the Security World. When performing a back-up of this data, you must include all the data in the `local` sub-directory, as described in the previous section.

Your Security World data is valuable. Access to the Security World folder should only be allowed for authorized users. Furthermore, it is possible to control usage permissions for individual cryptographic files to particular users only, in order to fine-grain authorized access to cryptographic operations where those files are used. However, check with your organisation's security policies before you do this.

If you need to set permissions to control access to individual cryptographic files, then you will need to know something about those files, as follows.

The following file is the minimum necessary data to initiate a functional Security World:

- `world` – holds information relating to the Security World's type, its other characteristics, and ACS cards.

The `world` file must be generated by the Security World software and is loaded onto the target HSM(s) upon creation. Otherwise, a pre-existing `world` file must be loaded onto the target HSM(s) using its ACS card(s). Please refer to the User Guide for your HSM for more information about creating or loading a Security World.

Unless the `world` file is loaded onto a usable HSM, no other cryptographic files associated with it in the `local` folder will function.

Other files in the `local` directory that may be associated with the `world` file are as follows. None of the these files can be created unless a `world` file already exists, and once they exist will only work with the `world` file they were created under.

- **`cards_<hash>`** - holds information about an OCS cardset where <hash> is a number unique to the cardset. The same <hash> will be used by all individual card files that are members of the same cardset.
- **`card_<hash>_<n>`** - holds information about an individual OCS card where <hash> is the OCS cardset hash, and <n> is the individual card's creation sequence number.
- **`softcard_<hash>`** - holds information about a softcard where <hash> is a number unique to the softcard.
- **`key_pkcs11_<hash>`** – holds information about a pkcs11 encryption key where <hash> is a number unique to the key. The SQL Server EKM API only works with pkcs11 keys.

Please refer to the *User Guide* for your HSM for more information about creating OCS or softcards.

All the above files are inherently encrypted and are useless to anyone who does not possess the correct authorizing mechanisms. Be very careful about deleting any of the above files from the `local` folder. Unless you have a back-up, any such file that is deleted from the `local` folder is lost for good.

The *local* subdirectory may also contain the following file(s), but which may not be needed if a different hardware configuration is used, and also should not be difficult to replace.

- **`module_<ESN>`** – where <ESN> is a module's Electronic Serial Number. Holds information about a HSM that is configured to use the Security World.

Together, the `world` file and the (above) files created under it comprise a Security World's cryptographic data. These files should always be kept exclusively together in their owning Security World folder. The contents of the Security World folder distinguish between different Security Worlds. Files from different Security World folders should **never** be mixed and will not work in the wrong Security World in any case (although keys can be imported using correct procedures).

Always make sure you have an up to date back-up of your Security World data that includes all files in the `local` folder.

Note: You can switch between different existing Security Worlds while retaining the same system configuration by renaming the desired Security World folder to `local`. You must then load the Security World by using its associated ACS cards.

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* on page 57.

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 back-up 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 (one only) 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 the 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.

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.

Refer to *Chapter 5: Security World Data and back-up/restore* on page 54 for information about locating the Security World data, and the files you need to back-up.

The Security World data is inherently 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 Thales 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.

Note: 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.

Backing up a database with SQL Server Management studio

Note: 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**.
   Note: 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 back up 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.
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.

Note: 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.

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.

Note: 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* on page 26 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 keys* on page 33 or *Re-importing an asymmetric key* on page 35.

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.

Chapter 6: Troubleshooting

(Relating to Thales SQLEKM provider only).

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)`	This usually indicates a problem with the pkcs11 configuration. Check if: - **`%NFAST_HOME%\toolkits\pkcs11`** is on the system PATH, and before installation of the SQLEKM provider. - The pkcs11 path is corrupted with wrong or stray characters. - 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.
When you attempt to create a key in the SQLEKM provider using the Microsoft SQL Server Management Studio, an error message similar to the following is returned - `Msg 33035, Level 16, State 1, Line 2 Cannot create key '<some_key_name>' in the provider. Provider error code: 1. (Failure - Consult EKM Provider for details)`	Using the Microsoft SQL Server Management Studio, try: Go to <*Database server name*> => Security => Cryptographic Providers => <*SQLEKM provider name*>. Right-click and select **Disable Provider**. Then, right-click and select **Enable Provider**. Wait for about a minute before repeating your attempt to create the key. If the above actions do not work, restart the MS SQL Server. (You may need administrator privileges to do this.)
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.

Problem / issue	Suggested diagnosis / solution
Microsoft SQL Server Management Studio displays a message stating that a DES key could not be created.	The DES key cannot be created because the Thales nShield HSM is operating at a strict level of compliance with the FIPS 140-2 Level 3 security standard. DES keys can only be created where the Thales nShield HSM is operating at a non-strict level of compliance.
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.
When you perform a query using the Microsoft SQL Server Management Studio, an error message similar to the following is returned - `Msg 10054, Level 20, State 0, Line 0 A transport-level error has occurred when sending the request to the server. (provider: TCP Provider, error: 0 - An existing connection was forcibly closed by the remote host.)`	This often means that some change occurred in the system where a communication channel was temporarily disrupted. Usually the channel will recover by itself. Wait a few moments and try the query again.
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.
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 *Appendix A: T-SQL shortcuts and tips* on page 67. • Update the database from the latest backup log.

Chapter 7: Uninstalling and Upgrading

CAUTION:

- 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.

Turning off TDE and removing TDE setup

You must turn off TDE on all your databases and remove TDE setup before uninstalling the Thales Database Security Option Pack for SQL Server. 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* on page 58) and associated Security World.

If you are using a version of SQL Server 2008, please read Microsoft SQL Server advice note Microsoft Article ID: 2300689. Check Microsoft support for details.

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 the section *How to check the TDE encryption/decryption state of a database* on page 41.
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
```

Uninstalling the Thales Database Security Option Pack for SQL Server

Do not uninstall the Thales Database Security Option Pack for SQL Server until you have:

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

To uninstall the Thales Database Security Option Pack for SQL Server from Microsoft SQL Server:

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 **Database Security Option Pack for SQL Server** then click **Uninstall**.
6. A dialog is displayed asking if you want to continue with uninstalling the Thales Database Security Option Pack for SQL Server. Click **Yes**.
7. A setup status screen is displayed while the Thales Database Security Option Pack for SQL Server is uninstalled. When InstallShield has finished uninstalling the program, 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**.

Upgrading

Enhancements will be made to the Thales Database Security Option Pack for SQL Server over time, and product upgrades made available to customers. To upgrade your product:

1. In SQL Server Management Studio, 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**.
2. Uninstall the existing Database Security Option Pack for SQL Server, using the procedure described in *Uninstalling the Thales Database Security Option Pack for SQL Server* on page 65.
3. Install the upgraded version of the Thales Database Security Option Pack for SQL Server, using the procedure described in *Installation* on page 17.
   Note: You must install the upgraded SQLEKM provider to the same directory as the previous installation. This ensures that the replacement provider files are found automatically when the Microsoft SQL Server instances are started.
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 > Start**.

Appendix A: 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.

Creating a database

To create a database called **TestDatabase**.

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

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
```

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]
```

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.

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).

Removing a credential

To remove a credential from the database:

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

See *Adding a credential* on page 69 for terms used.

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
```

Removing a TDEDEK

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

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

Switching on TDE

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

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

Switching off TDE

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

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

Dropping an 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.

Disabling SQLEKM Provision

To disable the EKM provision in an 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
```

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
```

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
```

Appendix B: Restarting a recovered HSM

In HSM loadsharing, where a HSM has failed but service has continued on a partner HSM, the SQLEKM provider will not automatically start using the failed HSM if it recovers and becomes available again.

The recommended procedure for restarting a recovered HSM:

- Close down all open sessions that are using the SQLEKM provider, disable and then re-enable the provider.
  This will cause SQL Server to re-initialize the SQLEKM provider, which will refresh the list of available HSMs.
  To do this in SQL Server Management Studio:
  - Close all open query windows.
  - In SQL Server Management Studio, navigate to **Security > Cryptographic Providers > Name of provider**.
  - Right-click **Name of provider** and choose **Disable Provider**.
  - Right click **Name of provider**, choose **Enable Provider**.

Alternatively, you can:

- Restart the SQL Server instance.
- If using a failover cluster, move the SQL Server instance to another node in the cluster.

Appendix C: Using TDE within 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. If you require different arrangements, please contact Thales support if you need assistance.

Note: If you have installed Thales V12.00 Security World software and you are using Java cards, be sure you have configured the cardlist file appropriately. In a cluster, you will need the same cardlist file contents on all servers in order to access the same cards. Please refer to the User Guide for your nShield HSM

Setting up and switching on TDE

Please note the following:

- The MSSQL Server Studio Add Database Wizard (versions to SQL Server 2014) will not support addition of a database that is already encrypted, or that includes a database encryption key even if encryption is switched off. However, you may set up TDE encryption for an existing non-encrypted database that is already within an availability group using T-SQL, as described below.
- SQL Server (versions to SQL Server 2014) may not support a readable secondary using a clustered columnstore index within the context of availability group failover. Please see https://connect.microsoft.com/SQLServer/feedback/details/1348268/availability-group-databasesnapshot-isolation-level-error-35371-on-readable-secondary.

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 for a log backup. In this case, please note section *Taking a log backup* on page 79.

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 Thales 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 Thales Security World and Database Security Option Pack for SQL Server software is installed and configured in the same manner as that described in the section *SQL Server database failover cluster using nShield Connects* on page 21 (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 the section *Enabling the SQLEKM provider* on page 25, 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. Note: We recommend a strong passphrase of at least 10 characters in length. Check your organisation'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, on 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 Sychronized', 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 section *Taking a log backup* on page 79. 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,
CASEe.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.

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';
```

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 the section *Setting up and switching on TDE* on page 73.

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 on page 78. 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. This appears to be a known Microsoft bug. 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.

Appendix D: Using an OCS quorum of K/N where K>1

Note: In the SQLEKM context, using an OCS quorum where K>1 **is not recommended**, as explained below. For this reason, it has only been tested in a SQLEKM context for a standalone system using one HSM module, and this is the only configuration described here. Both Cell and TDE encryption were tested to work satisfactorily in this standalone configuration.

Overview

A SQL Server credential (as used for EKM) maps one protecting token that is an OCS card or softcard, to one stored passphrase. Softcards are singular and do not have a quorum, so the SQL Server credential matches them quite well. On the other hand, an OCS cardset does have a quorum, but as the SQL Server credential can store information for only one token at a time, a quorum greater than one cannot be directly supported. Neither can different passphrases for each card in an OCS cardset be supported by the same credential.

Furthermore, a SQL Server user login can only be associated with one SQL Server credential at a time (although a credential can be associated with more than one login at the same time). Therefore direct use of a SQL Server credential implies that we are restricted to using an OCS cardset with a 1/N quorum, and every OCS card must use an identical passphrase if they are to be used interchangeably with the same credential. But this means the benefits of a quorum of more than one OCS card are lost.

As will be shown below, it is possible to use a quorum of multiple OCS cards (K/N where K>1) by preloading. Unfortunately, this opens up extra security risks and failure recovery problems. Its use is not normally recommended in a SQLEKM context.

In order to employ a quorum of more than one OCS card, we must use the Thales 'preload' utility to load the OCS quorum onto a nShield HSM before use. This will also require setting up a tokens file. Preloading allows an SQL Server credential to function with a designated OCS (K>1) cardset for an associated user login.

Obviously, for the 'preload' utility to run, it must be in a suitably installed configuration as described elsewhere in this document, with a usable Security World and HSM module available. The OCS cardset with K>1 must have already been created. In this case, each OCS card can have a different passphrase.

Using the preload utility

To use the Thales **`preload`** utility with the SQLEKM provider, you must:

1. Set up a path to a `tokens file` using the *NFAST_NFKM_TOKENSFILE* environment variable. That is, **`NFAST_NFKM_TOKENSFILE=<path>\<name-of-tokens-file`**.

   If a tokens file of the same name already exists at the location given, delete it.

For security reasons, you must take care as to who can access the tokens file, and protect it. If there is only one, or a small number of logins that need to use this variable, it may be practicable to set up *NFAST_NFKM_TOKENSFILE* as a local variable for each user. Otherwise, set it up as a system variable, which is likely to be more suitable if you are using TDE. For the subdirectory that will contain the tokens file, either:

- Set read/write/execute permissions for each individual (login) that needs to use the tokens file, or
- Create a new user group, and provide group read/write/execute permissions for members of that group to use the tokens file.
- All other users should be excluded access to the tokens file.

2. Run the **preload** utility on a command line where *<OCS-name>* is the name of an OCS cardset in which K>1.

```
>> preload -f "%NFAST_NFKM_TOKENSFILE%" --cardset-name=<OCS-name> pause
```

The utility will prompt you to insert a succession of OCS cards into the HSM card reader, and enter their correct passphrase, until the quorum is reached. Do not remove the final card from the HSM slot. When the quorum is complete, the preload utility goes into a paused state. Do not terminate it.

3. Restart the SQL server.
4. Set up a SQL Server credential for the OCS name. In this case, while an OCS passphrase must be supplied for the credential, it will be ignored in practice. Associate the credential with the required login. If setting up a TDE, the user must utilize the same OCS for their login credential as used for the TDE credential. For a standalone configuration, create the TDEKEK, TDE login and credential, TDEDEK, and switch on encryption in the usual way.

Example for standalone system

In this example the OCS cardset used is **OCS3** and it has a 2/3 quorum:

1. Select **Start** => **Control Panel\System and Security\System\Advanced System Setting\Environment Variables** to set up *NFAST_NFKM_TOKENSFILE*.

   Provide suitable protections for access to the tokens file. For example:

```
NFAST_NFKM_TOKENSFILE=C:\ProgramData\nCipher\Key Management Data\TestPreload\Tokensfile
```

   If a tokens file of the same name already exists at the location given, delete it.

2.
```
>> preload -f "%NFAST_NFKM_TOKENSFILE%" --cardset-name=OCS3 pause

Loading cardsets:
OCS3 on modules 1

Loading `OCS3':
Module 1 slot 0: `OCS3' #2
You must enter a passphrase for this card.
Module 1 slot 0:- passphrase supplied - reading card
Module 1 slot 0: `OCS3' #2: already read
Module 1 slot 0: empty
Module 1 slot 0: `OCS3' #1
You must enter a passphrase for this card.
Module 1 slot 0:- passphrase supplied - reading card
Card reading complete.


Stored Cardset: OCS3 (ce63...) on module #1
Loading complete; now pausing

[Do not terminate the preload program.]
[Do not remove final card of quorum from HSM slot.]
```

3. Restart SQL Server.
4. Set up a SQL Server credential for the OCS cardset (in this case, the passphrase supplied will be ignored), and associate the credential with the required login(s).
5. You should now be able to run queries where the SQLEKM provider keys are protected by the K/N cardset where K>1.

Operational considerations

After running the **preload** utility as shown above, it will create the tokens file in the specified folder. At this point the OCS card authorization should have been set up in the HSM module. Check that the authorization works by performing some queries that require encryption keys protected by the OCS cardset. If the authorization works, then:

- If the OCS cardset is persistent, you may remove the final card from the HSM slot
- If the OCS cardset is not persistent, you must leave the final card in the HSM slot, as removing it will lose the cardset authorization.

The **preload** utility creates the tokens file. This file holds logical IDs or tokens that are internal to the Thales software, and relate to the OCS cardset that is loaded, and to any keys the cardset protects. These tokens are used by the Thales hardserver and HSM modules within the configuration to provide the OCS card authorization. Unless there is some system failure, the authorization remains valid until the hardserver is restarted, or the module(s) are cleared. Note too, that if the OCS cardset is non-persistent, then removing the final card of the quorum from the HSM slot will also lose the authorization.

Each time a new tokens file is created, a different set of logical tokens may be generated, even if the same OCS card(s) are being loaded. A tokens file cannot be reused once its current authorization is lost. You should delete a tokens file that was previously used before creating a new one to replace it.

Please note that once the tokens file has been created and whilst its authorization remains valid, the OCS cards it has loaded, and by implication any keys they are protecting, are available to any user or application that can access that file and invoke the Thales software.

Unless care is taken, authorization to use the OCS cardset might not be tied to specific users or applications. Also be aware that the passphrase supplied in the SQL Server credential is ignored, although it is a stored password in any case. These factors represent a security risk. Steps must therefore be taken to restrict access to the tokens file and Thales software, so that only the correct users or applications can use them.

In the event that there is a failure of a HSM module or hardserver, say due to a temporary power outage, the preloaded authorization will be lost. When the system returns to operation, the tokens file will be invalid and a new one will have to be created. This will require deletion of the previous tokens file, a repeat of the preload command, and manual insertion of the quorum of OCS cards. In other words, the system cannot be set up to recover automatically.

Within a SQLEKM context, because of the extra security risks and poor failure recovery characteristics, we recommend that you do not use a preloaded cardset. By implication, this means you should not use a K/N OCS cardset where K>1, unless you have a strong reason for doing so. It may possibly be acceptable for a small private system with restricted usage, which is closely monitored, and where manual recovery from failure is tolerable. For a large or public system that is in continuous use, and for which automatic recovery from failover is desired, then we do **not** recommend you use this method.

Internet addresses

Web site:	http://www.thales-esecurity.com/
Support:	http://www.thales-esecurity.com/support-landing-page
Online documentation:	http://www.thales-esecurity.com/knowledge-base
International sales offices:	http://www.thales-esecurity.com/contact

Addresses and contact information for the main Thales e-Security sales offices are provided at the bottom of the following page.

THALES

www.thales-esecurity.com

About Thales e-Security

Thales e-Security is a leading global provider of data encryption and cyber security solutions to the financial services, high technology manufacturing, government and technology sectors. With a 40-year track record of protecting corporate and government information, Thales solutions are used by four of the five largest energy and aerospace companies, 22 NATO countries, and they secure more than 80 percent of worldwide payment transactions. Thales e-Security has offices in Australia, France, Hong Kong, Norway, United Kingdom and United States. For more information, visit www.thales-esecurity.com

Follow us on:

Americas – Thales e-Security Inc. 900 South Pine Island Road, Suite 710, Plantation, FL 33324 USA • Tel: +1 888 744 4976 or +1 954 888 6200 • Fax: +1 954 888 6211 • E-mail: sales@thalesesec.com
Asia Pacific – Unit 4101 41/F 248, Queen's Road East, Wanchai, Hong Kong, PRC • Tel: +852 2815 8633 • Fax: +852 2815 8141 • E-mail: asia.sales@thales-esecurity.com
Europe, Middle East, Africa – Meadow View House, Long Crendon, Aylesbury, Buckinghamshire HP18 9EQ • Tel: +44 (0)1844 201800 • Fax: +44 (0)1844 208550 • E-mail: emea.sales@thales-esecurity.com