Microsoft
SQLEKM CP5 Windows Server

Integration Guide

CryptoServer

utimaco®

Imprint

Copyright 2020	Utimaco IS GmbH
	Germanusstr. 4
	D-52080 Aachen
	Germany
Phone	AMERICAS +1-844-UTIMACO (+1 844-884-6226)
	EMEA +49 800-627-3081
	APAC +81 800-919-1301
Internet	https://support.hsm.utimaco.com/
e-mail	support@utimaco.com
Document Version	1.0.0
Date	06/10/2025
Status	PUBLISHED
Document No.	IG-2025-0007

Table of Contents

1 Overview

This document describes how to setup and use the Utimaco CryptoServer SQLEKM provider. The Microsoft SQL Server provides data encryption, decryption and key management capabilities. Together with the Extensible Key Management (EKM) the management of encryption keys for data and key encryption is very easy. Since Microsoft SQL Server 2008 (R2), Extensible Key Management enables third-party EKM vendors like Utimaco to register their EKM modules within Microsoft SQL Server. When registered, Microsoft SQL Server users can use the encryption keys stored on the Utimaco CryptoServer hardware security module. This enables Microsoft SQL Server to access the advanced encryption and protection features of the Utimaco CryptoServer.

2 Integration Details

2.1 Tested Versions

Operating System	SecurityServer Release	SQL Server
Windows Server 2016	4.30 4.31	SQL Server 2017 Enterprise
Windows Server 2016	4.10	SQL Server 2014 Enterprise SQL Server 2016 Enterprise SQL Server 2017 Enterprise
Windows Server 2012 R2	4.10	SQL Server 2012 Enterprise SQL Server 2014 Enterprise SQL Server 2016 Enterprise SQL Server 2017 Enterprise

Tested Versions

Note that this list is not exclusive and other combinations of the versions tested above should work as well.

In SecurityServer Release 4.20 a wrong EKM provider DLL was delivered and, therefore, that release is not working.

The Enterprise version of SQL Server is required to use the EKM provider.

2.2 Supported Algorithms

Algorithm	Algorithm Tag
DES	DES

Algorithm	Algorithm Tag
Triple DES with 128 bit key	TRIPLE_DES
Triple DES with 168 bit key	TRIPLE_DES_3KEY
AES 128 bit	AES_128
AES 192 bit	AES_192
AES 256 bit	AES_256
RSA 512 bit	RSA_512
RSA 1024 bit	RSA_1024
RSA 2048 bit	RSA_2048
RSA 3072 bit	RSA_3072
RSA 4096 bit	RSA_4096

RSA_3072 and RSA_4096 are supported starting with SecurityServer release 4.31.

3 Prerequisites

Before you begin, please ensure that you have installed/setup:

- Operating system listed in Tested Versions
- SQL Server listed in Tested Versions
- SQL Server Management Studio
- SecurityServer listed in Tested Version with CryptoServer SQLEKM provider
- CryptoServer (PCIe or LAN) with MBK loaded
- A cryptographic user on that CryptoServer

You should also be familiar with SQL statements, as this guide makes intensive use of them.

For more information consult the CryptoServer Manual for System Administrators.

After the successful SecurityServer setup, you should find these files on your system for use with the CryptoServer SQLEKM provider:

- `cssqlekm.dll and cssqlekmlib.dll`

  The former is the provider library that will be loaded into SQL Server, and the latter is required by
  it. These files are located in `C:\Program Files\Utimaco\CryptoServer\Lib\`, which must be in system `PATH`.

- `cssqlekm.cfg`
  This file contains the parameters that the CryptoServer SQLEKM provider will use when communicating with the HSM. Please see the next sections for details.

Before SecurityServer release 4.30 the location of the EKM DLLs was not added to the system `PATH` if only the EKM API was selected in the installation wizard. Workaround: Also install the PKCS#11 API or add the location manually. Restart the computer afterwards.

3.1 Configuration File

3.2 Location of the Configuration File

The installation wizard copies a sample configuration file to `C:\ProgramData\Utimaco\EKM\cssqlekm.cfg`. Please see the next section on how to customize the file.

Starting with SecurityServer 4.31, the installation wizard also adds the ConfigPath value to the Windows registry key `HKEY_LOCAL_MACHINE\SOFTWARE\Utimaco\EKM` containing the location of the configuration file.

Alternatively it is possible to register the location of the configuration file using an environment variable, which takes precedence over the registry setting: Open the Control Panel, System, click on Advanced system settings and click on the Environment Variables. Create a new variable `EKMCONFIGPATH` in System variables. Add the path of `cssqlekm.cfg`. A restart is necessary.

Before SecurityServer release 4.31 setting the environment variable was the only option to specify the location of the configuration file. Before SecurityServer release 4.21 the location of the configuration file was `C:\Program Files\Utimaco\CryptoServer\Software\EKM\cssqlekm.cfg`.

3.3 Customization of the Configuration File

The CryptoServer SQLEKM provider can be customized using the configuration file. Edit the configuration file to your needs. At least, change the `Device` setting. Make sure the MSSQL Server service account has write access to the folder containing the external key store and the log file.

Parameter	Description
LogFile	Specifies the path and the name of the log file.

Parameter	Description
LogLevel	Specifies the log level. Higher levels include the information of the lower levels. 0=no log 1=errors 2=warnings 3=info 4=trace 5=debug.
LogSize	Defines the maximum size of the log file. If the maximum is reached, the old log file will be renamed to `.bak` and a new log file with the name defined by `LogFile` is created.
KeyStore	Specifies the path and the name of the external key store file. Note that currently no internal key store is supported.
Device	Specifies the device address of the CryptoServer device. This can be a local PCIe card (`PCI:0`) or a network address (`[host@]IP`).
ConnectionTimeout	Specifies the maximum time in milliseconds to wait before the connection establishment is aborted if the device is not responding.
Timeout	Specifies the maximum time in milliseconds to wait for the answer from CryptoServer after sending a command.

3.4 IG-2025-0007 Setting up Credentials

The CryptoServer SQLEKM provider exposes basic authentication to the SQL Server using username/- password pairs. These pairs are stored in so-called credentials that need to be created per EKM provider. Finally, a credential is mapped to an SQL server login.

If a logged in user wants to access a certain EKM provider the credential mapped to both the login and the EKM provider is looked up and the username/password is passed to the EKM provider. The CryptoServer SQLEKM provider uses this information to perform login on the CryptoServer.

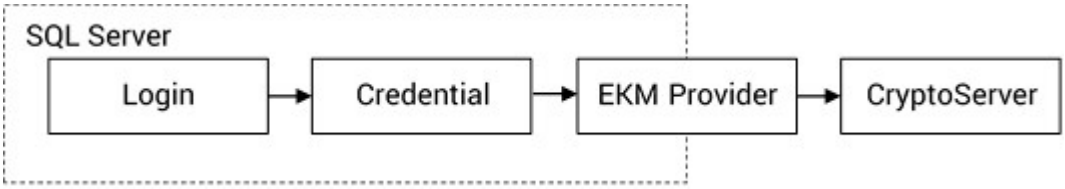


Figure 1 : Credential mapping

The same credential can be used for multiple SQL server logins. Also, a login can be used with multiple credentials as long as the EKM providers are different. Otherwise the lookup shown before will fail.

The following SQL will create a credential

csekm

for the CryptoServer user

sqlekm

with the

password

utimaco

.

SQL Statement

CREATE

CREDENTIAL csekm

WITH

IDENTITY =

'sqlekm'

, SECRET =

'utimaco'

FOR

CRYPTOGRAPHIC PROVIDER utimaco

Creating a CryptoServer user need to be done either via csadm or the CAT Administration. For detailed information refer to chapter 4.14.1 of the CryptoServer Manual Systemadministrator.

Use the following SQL statement to map the credential to any SQL Server account. You can for example substitute <user> with an integrated account like sa or an Windows account like [DB1\Administrator] .

SQL Statement

ALTER LOGIN <user> ADD CREDENTIAL csekm

3.4.1 IG-2025-0007 Improving Security via CXI Group

By default, new EKM keys are generated without CXI group. The CryptoServer user does not need to have a CXI_GROUP attribute, but every cryptographic user on the CryptoServer can

Installation

access the keys in the SQLEKM key store file. To provide better protection, a CXI group should be defined in the SQL Server credential's identity:

SQL Statement

CREATE CREDENTIAL csekm WITH IDENTITY = 'sqlekm@ekmgroup', SECRET =

'utimaco'

FOR CRYPTOGRAPHIC PROVIDER utimaco

Now, new SQLEKM keys are created in the CXI group ekmgroup and only CryptoServer users belonging to this group can access these keys. Therefore, the CryptoServer user sqlekm nee ds to be member of the CXI group ekmgroup by setting its CXI_GROUP attribute to

ekmgroup on user creation.

Since key names (more specifically the PROVIDER_KEY_NAME) have to be unique per CXI group only, the use of different CXI groups for different credentials also prevents name collisions when SQLEKM is used with different databases from the same SQL Server.

3.4.2 IG-2025-0007 Cryptographic User Hierarchy

Since SecurityServer release 4.31 the CryptoServer SQLEKM provider also supports hierarchical users via wildcards in the CXI_GROUP user attribute. Imagine the following SQL Server credentials with their identities and the CXI_GROUP attribute of the matching CryptoServer users:

3.4.2.1 Credential Identity

EKMuser1 EKMuser1@ekmgrp1 ekmgrp1

EKMuser2 EKMuser2@ekmgrp2 ekmgrp2 EKMadmin EKMadmin@ekmgrp1 ekmgrp*

Table 3: Example SQL server credentials and identities

An SQL Server account bound to credential EKMuser1 is logged into CryptoServer as

EKMuser1 . New keys are generated in CXI group ekmgrp1 and only keys in that group can be accessed. Similarly, an SQL Server account bound to credential EKMuser2 is logged in as user EKMuser2 , and new keys are generated in CXI group ekmgrp2 . Unsurprisingly, an account bound to credential EKMadmin is logged into CryptoServer as EKMadmin . New keys are now generated in CXI group ekmgrp1 since this value is taken from the credential's identity. However, this user can also use keys generated by user EKMuser2 in group ekmgrp2 since the CXI_GROUP attribute grants access to these keys as well. This works since SQL Server commands refer to a key by an SQL Server name, which is bound internally to an EKM provider key name defined in the CREATE … KEY statement, and the actual access is done using an identifier stored together with the SQL Server name. This identifier is also used when the key is deleted inside the provider.

Note that the CXI group from the credential's identity is also used when a key inside the CryptoServer is searched by name. This happens when a new SQL Server key is created from an existing CryptoServer key. Currently, supplying a different CXI group in the CREATE … KEY

statement than the one given in the credential is not supported. Moreover, there would be no way to explicitly specify the CXI group when viewing all the keys from the SQLEKM provider.

Using the Provider

4 Installation

4.1 Enable Extensible Key Management

The `EKM provider enabled` option controls Extensible Key Management device support in Microsoft SQL Server. This option is disabled by default and needs to be enabled in order to use any EKM provider. Connect to your SQL Server instance and login with administrative privileges. Open a query window for further execution of SQL statements. To enable Extensible Key Management, please run the following SQL.

SQL Statement

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

4.2 Register Provider

Run the following SQL to register the provider under the name `utimaco`.

SQL Statement

```
CREATE CRYPTOGRAPHIC PROVIDER utimaco
FROM FILE = 'C:\Program Files\Utimaco\CryptoServer\Lib\cssqlekm.dll'
```

To verify the successful installation, run:

SQL Statement

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

Make sure there is a *Utimaco CryptoServer SQLEKM Provider* registered.

4.2.1 Alter Provider Location

To change the location of the provider, run the following at any time.

SQL Statement

```
ALTER CRYPTOGRAPHIC PROVIDER utimaco
FROM FILE = '<path-to-new-provider-dll>'
```

Since SQL Server stores the version of an EKM provider with the registration, it is also necessary to run this command after updating the provider.

4.2.2 Remove Provider

To remove the provider entirely:

1. Close all opened sessions that use the provider.
2. Remove all credentials regarding the provider.
3. Run the following SQL statement.

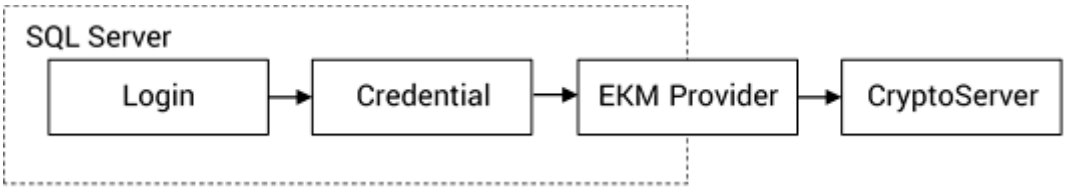
   SQL Statement

   ```
   DROP CRYPTOGRAPHIC PROVIDER utimaco
   ```

4.3 Setting up Credentials

The CryptoServer SQLEKM provider exposes basic authentication to the SQL Server using username/- password pairs. These pairs are stored in so-called credentials that need to be created per EKM provider. Finally, a credential is mapped to an SQL server login.

If a logged in user wants to access a certain EKM provider the credential mapped to both the login and the EKM provider is looked up and the username/password is passed to the EKM provider. The CryptoServer SQLEKM provider uses this information to perform login on the CryptoServer.



Credential mapping

The same credential can be used for multiple SQL server logins. Also, a login can be used with multiple credentials as long as the EKM providers are different. Otherwise the lookup shown before will fail.

The following SQL will create a credential `csekm` for the CryptoServer user `sqlekm` with the password `utimaco`.

SQL Statement

```
CREATE CREDENTIAL csekm WITH IDENTITY = 'sqlekm', SECRET = 'utimaco'
FOR CRYPTOGRAPHIC PROVIDER utimaco
```

Creating a CryptoServer user need to be done either via csadm or the CAT Administration. For detailed information refer to chapter 4.14.1 of the CryptoServer Manual Systemadministrator.

Use the following SQL statement to map the credential to any SQL Server account. You can for example substitute `<user>` with an integrated account like `sa` or an Windows account like `[DB1\Administrator]`.

SQL Statement

```
ALTER LOGIN <user> ADD CREDENTIAL csekm
```

4.3.1 Improving Security via CXI Group

By default, new EKM keys are generated without CXI group. The CryptoServer user does not need to have a `CXI_GROUP` attribute, but every cryptographic user on the CryptoServer can access the keys in the SQLEKM key store file. To provide better protection, a CXI group should be defined in the SQL Server credential's identity:

SQL Statement

```
CREATE CREDENTIAL csekm WITH IDENTITY = 'sqlekm@ekmgroup', SECRET =
'utimaco'
FOR CRYPTOGRAPHIC PROVIDER utimaco
```

Now, new SQLEKM keys are created in the CXI group `ekmgroup` and only CryptoServer users belonging to this group can access these keys. Therefore, the CryptoServer user `sqlekm` needs to be member of the CXI group `ekmgroup` by setting its CXI_GROUP attribute to `ekmgroup` on user creation.

Since key names (more specifically the `PROVIDER_KEY_NAME`) have to be unique per CXI group only, the use of different CXI groups for different credentials also prevents name collisions when SQLEKM is used with different databases from the same SQL Server.

4.3.2 Cryptographic User Hierarchy

Since SecurityServer release 4.31 the CryptoServer SQLEKM provider also supports hierarchical users via wildcards in the `CXI_GROUP` user attribute. Imagine the following SQL Server credentials with their identities and the `CXI_GROUP` attribute of the matching CryptoServer users:

Credential	Identity	
EKMuser1	EKMuser1@ekmgrp1	ekmgrp1
EKMuser2	EKMuser2@ekmgrp2	ekmgrp2
EKMadmin	EKMadmin@ekmgrp1	ekmgrp*

Example SQL server credentials and identities

An SQL Server account bound to credential `EKMuser1` is logged into CryptoServer as `EKMuser1`. New keys are generated in CXI group ekmgrp1 and only keys in that group can be accessed. Similarly, an SQL Server account bound to credential `EKMuser2` is logged in as user `EKMuser2`, and new keys are generated in CXI group `ekmgrp2`. Unsurprisingly, an account bound to credential `EKMadmin` is logged into CryptoServer as `EKMadmin`. New keys are now generated in CXI group `ekmgrp1` since this value is taken from the credential's identity. However, this user can also use keys generated by user `EKMuser2` in group ekmgrp2 since the `CXI_GROUP` attribut e grants access to these keys as well. This works since SQL Server commands refer to a key by an SQL Server name, which is bound internally to an EKM provider key name defined in the `CREATE … KEY` statement, and the actual access is done using an identifier stored together with the SQL Server name. This identifier is also used when the key is deleted inside the provider.

Note that the CXI group from the credential's identity is also used when a key inside the CryptoServer is searched by name. This happens when a new SQL Server key is created from an existing CryptoServer key. Currently, supplying a different CXI group in the `CREATE … KEY` state ment than the one given in the credential is not supported. Moreover, there would be no way to explicitly specify the CXI group when viewing all the keys from the SQLEKM provider.

5 Using the Provider

SQL Server can create and store keys internally, protected by software only. With Extensible Key Management (EKM), SQL Server can use keys protected by an HSM for data and key encryption/ decryption.

The CryptoServer SQLEKM provider offers EKM functionality for Utimaco CryptoServer HSMs, supporting different symmetric and asymmetric algorithms. Keys generated with the CryptoServer SQLEKM provider are stored in an external database ("keystore") encrypted by the Master Backup Key (MBK). The location of the database is defined in the configuration file.

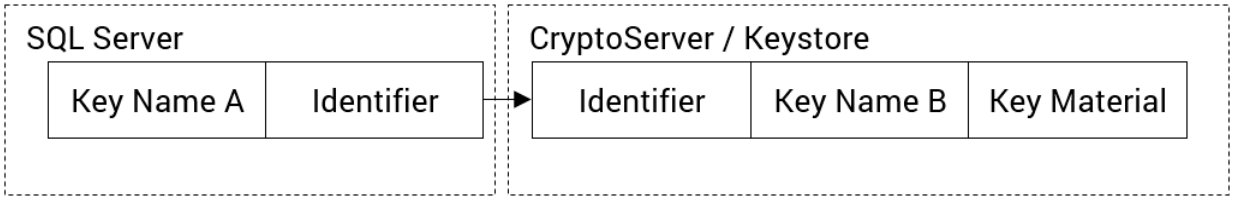
5.1 Creating Keys

To create a new symmetric AES 256 key `EKM_AES_256` and store it in the CryptoServer SQLEKM provider, use the following statement:

SQL Statement

```
CREATE SYMMETRIC KEY EKM_AES_256
FROM PROVIDER utimaco
WITH ALGORITHM = AES_256,
PROVIDER_KEY_NAME = 'EKM_AES_256',
CREATION_DISPOSITION=CREATE_NEW
```

Note that the key name `EKM_AES_256` appears twice here: first as key name for the SQL Server and second as the CryptoServer key name. However, it is not necessary that both names are identical. In fact, in SQL Server commands a key is referred to by its SQL Server name. The `CREATE … KEY` statement creates a binding to the CryptoServer key, which can be different, using a common identifier.



Key mapping

An SQL Server key can also be created from an existing CryptoServer SQLEKM provider key:

SQL Statement

```
CREATE SYMMETRIC KEY EKM_AES_256
FROM PROVIDER utimaco
WITH PROVIDER_KEY_NAME = 'OtherAesKey',
CREATION_DISPOSITION=OPEN_EXISTING
```

Here, a lookup for the given provider key name is performed. For the CryptoServer SQLEKM provider, the `CXI_GROUP` is also taken into account if one is specified in the credential's identity (see 2021-0004 Setting up Credentials). This statement creates the aforementioned binding.

To create asymmetric keys proceed in the same manner. Here is the statement to create an asymmetric RSA 2048 key:

SQL Statement

```
CREATE ASYMMETRIC KEY EKM_RSA_2048
FROM PROVIDER utimaco
WITH ALGORITHM = RSA_2048,
PROVIDER_KEY_NAME = 'EKM_RSA_2048',
CREATION_DISPOSITION=CREATE_NEW
```

5.2 Viewing Keys

The SQL Server provides several SQL statements to list existing cryptographic keys. The first set of statements is used to show the SQL Server keys, both keys stored internally and keys with references to EKM providers. This key listing is separated into symmetric and asymmetric listings.

Use the next SQL statement to show current symmetric keys:

SQL Statement

```
SELECT * from master.sys.symmetric_keys
```

This will show a listing of current asymmetric keys:

SQL Statement

```
SELECT * from master.sys.asymmetric_keys
```

Remember that only these keys can be used in SQL statements.

Additionally, all keys stored in the CryptoServer SQLEKM provider can be listed with an extra SQL statement. These keys can be used in a `CREATE … KEY` statement as `PROVIDER_KEY_NAME` to create an SQL Server key binding.

SQL Statement

```
SELECT * FROM sys.dm_cryptographic_provider_keys(65536)
```

Note that the number of shown keys can differ between previous mentioned SQL statements since not all keys stored in CryptoServer SQLEKM provider necessarily have an equivalent key in the SQL Server space and vice versa.

5.3 Deleting Keys

To delete an symmetric key in SQL Server use this SQL statement:

SQL Statement

```
DROP SYMMETRIC KEY <key name>
```

For asymmetric keys use `ASYMMETRIC` instead of `SYMMETRIC` in the SQL statement:

SQL Statement

```
DROP ASYMMETRIC KEY <key name>
```

With the previous statements an internal SQL Server key or a binding to a key in an EKM provider is deleted. In the latter case, the key itself is still existing in the EKM provider. To delete both the binding and the EKM provider key use the following statement for a symmetric key:

SQL Statement

```
DROP SYMMETRIC <key name> REMOVE PROVIDER KEY
```

For example:

SQL Statement

```
DROP SYMMETRIC KEY EKM_AES_256 REMOVE PROVIDER KEY
```

Use this SQL statement to delete an asymmetric key.

SQL Statement

```
DROP ASYMMETRIC KEY <key name> REMOVE PROVIDER KEY
```

For example:

SQL Statement

```
DROP ASYMMETRIC KEY EKM_RSA_2048 REMOVE PROVIDER KEY
```

6 Column Level Encryption

An EKM provider can be used for column-level encryption and decryption. This chapter shows how to use the CryptoServer SQLEKM provider as a column-level encryption and decryption engine. To demonstrate encryption and decryption a table demo will be created first. Take into account here that in order for cryptographic keys to be successfully used, they have to be generated within the same database as the table entries which you wish to encrypt.

SQL Statement

```
CREATE DATABASE utimaco
GO
USE utimaco
CREATE TABLE [dbo].[demo] (
firstname varchar (255) NOT NULL,
name varchar (255) NOT NULL,
secret varbinary (8000) NOT NULL
)
GO
CREATE SYMMETRIC KEY CLE_AES_256
FROM PROVIDER utimaco
WITH ALGORITHM = AES_256,
PROVIDER_KEY_NAME = 'CLE_AES_256',
CREATION_DISPOSITION=CREATE_NEW
GO
```

New rows can be inserted like in the next SQL statement. This statement uses a symmetric column encryption for the column secret.

SQL Statement

```
INSERT INTO demo
VALUES ('John', 'Doe', ENCRYPTBYKEY(KEY_GUID('CLE_AES_256'), 'utimaco'))
```

In the same way an asymmetric encryption could be used.

SQL Statement

```
INSERT INTO demo
VALUES ('John', 'Doe', ENCRYPTBYASYMKEY(ASYMKEY_ID('CLE_RSA_2048'),
'utimaco'))
```

EncryptByAsymKey returns NULL if the input exceeds a certain number of bytes, depending on the algorithm. The limits are:

- a 512 bit RSA key can encrypt up to 53 bytes
- a 1024 bit key can encrypt up to 117 bytes
- a 2048 bit key can encrypt up to 245 bytes

Encryption and decryption with an asymmetric key is very costly compared with encryption and decryption with a symmetric key.

To show the decrypted value of an encrypted column the next statements can be used. This decrypts the symmetric encrypted column address and shows all stored rows of this table:

SQL Statement

```
SELECT CONVERT(varchar, DECRYPTBYKEY(secret)) secret FROM demo
```

A decryption of asymmetric column can be achieved similar to the decryption of symmetric encrypted column:

SQL Statement

```
SELECT
CONVERT(varchar, DECRYPTBYASYMKEY(ASYMKEY_ID('CLE_RSA_2048'), secret))
secret
FROM demo
```

7 Transparent Data Encryption

With the introduction of transparent data encryption (TDE) in SQL Server 2008, users now have the opportunity of full database-level encryption by using TDE. TDE is the optimal choice for bulk encryption to meet regulatory compliance or corporate data security standards. TDE works at the file level which encrypts data directly on the hard drive. TDE does not replace the column-level encryption. It is just another way of encrypting data of your database transparently. The next steps will guide you on how to enable TDE with the CryptoServer SQLEKM provider.

First of all, a credential for TDE has to be created (see 2021-0004 Setting up Credentials).

SQL Statement

```
CREATE CREDENTIAL tde WITH IDENTITY = 'tde', SECRET = 'utimaco'
FOR CRYPTOGRAPHIC PROVIDER utimaco
```

Create an asymmetric key used as TDE KEK (Key Encryption Key) in the master database (see 2021-0004 Creating Keys).

SQL Statement

```
USE master;
CREATE ASYMMETRIC KEY tdekey
FROM PROVIDER utimaco
WITH ALGORITHM = RSA_2048,
PROVIDER_KEY_NAME = 'tdekey',
CREATION_DISPOSITION=CREATE_NEW;
```

Create a SQL Server login account from this asymmetric key:

SQL Statement

```
CREATE LOGIN tdelogin FROM ASYMMETRIC KEY tdekey
```

Link your SQL Server credential to your just created user account with the next statement:

SQL Statement

```
ALTER LOGIN tdelogin ADD CREDENTIAL tde
```

Switch to your database to be encrypted with TDE. In our example we create a database named demo first:

SQL Statement

```
CREATE DATABASE demo
GO


USE demo
```

Create a database encryption key, in this example based on an AES algorithm.

SQL Statement

```
CREATE DATABASE ENCRYPTION KEY
WITH ALGORITHM = AES_256
ENCRYPTION BY SERVER ASYMMETRIC KEY tdekey
```

Enable the transparent data encryption and start encryption of the database as a background thread. Depending on the size of the database it can take a while until the encryption has been completed.

SQL Statement

```
ALTER DATABASE demo SET ENCRYPTION ON;
```

To see the current state of the encryption, use the next SQL statement.

SQL Statement

```
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.asymmetric_keys AS c
ON e.encryptor_thumbprint = c.thumbprint;
Page
```

8 Troubleshooting

The SQLEKM provider reports problems in two ways:

1. Until the initialization of the EKM provider has finished errors are written to the Windows Event Log. Open the Event Viewer on Windows and check the Application log for messages from event source "Utimaco EKM Provider". Typical errors are that the path to the configuration file is not defined, that the configuration file cannot be read, that the key store file cannot be opened, or that the log file could not be created/opened.

2. After the initialization, the log file is used for reporting. The log file and the log level need to be set in the configuration file. Make sure to have at least log level 1 to see errors. Before contacting the support try to reproduce the error with log level set to 5. After changing the log level you need to restart the SQL Server service.

9 Further Reading

For in-depth information on EKM or related topics, please refer to the Microsoft Technet.

https://docs.microsoft.com/en-us/sql/relational-databases/security/encryption/extensible-key-management-ekm

10 Contact Address for Support Queries

You can reach us from Monday to Friday, 09.00 a.m. to 05.00 p.m., Central European Time (CET).

Utimaco IS GmbH
Germanusstr. 4
52080 Aachen
Germany

RMA Query

If you need to send the device back to Utimaco IS GmbH, please open a new RMA case (Return Merchandise Authorization). We request that you use the following web address. RMA cases cannot be opened by email or phone.

https://support.hsm.utimaco.com/support/rma/new

Other Support Queries

- Mail (preferred contact method)
  support@utimaco.com
  Attach the diagnostic information to your email.
- Web portal
  https://support.hsm.utimaco.com/support/cases/new/
  The diagnostic information will be requested in our response if necessary.
- By phone
  AMERICAS +1-844-UTIMACO (+1 844-884-6226)
  EMEA +49 800-627-3081
  APAC +81 800-919-1301
  The diagnostic information will be requested in our response if necessary.