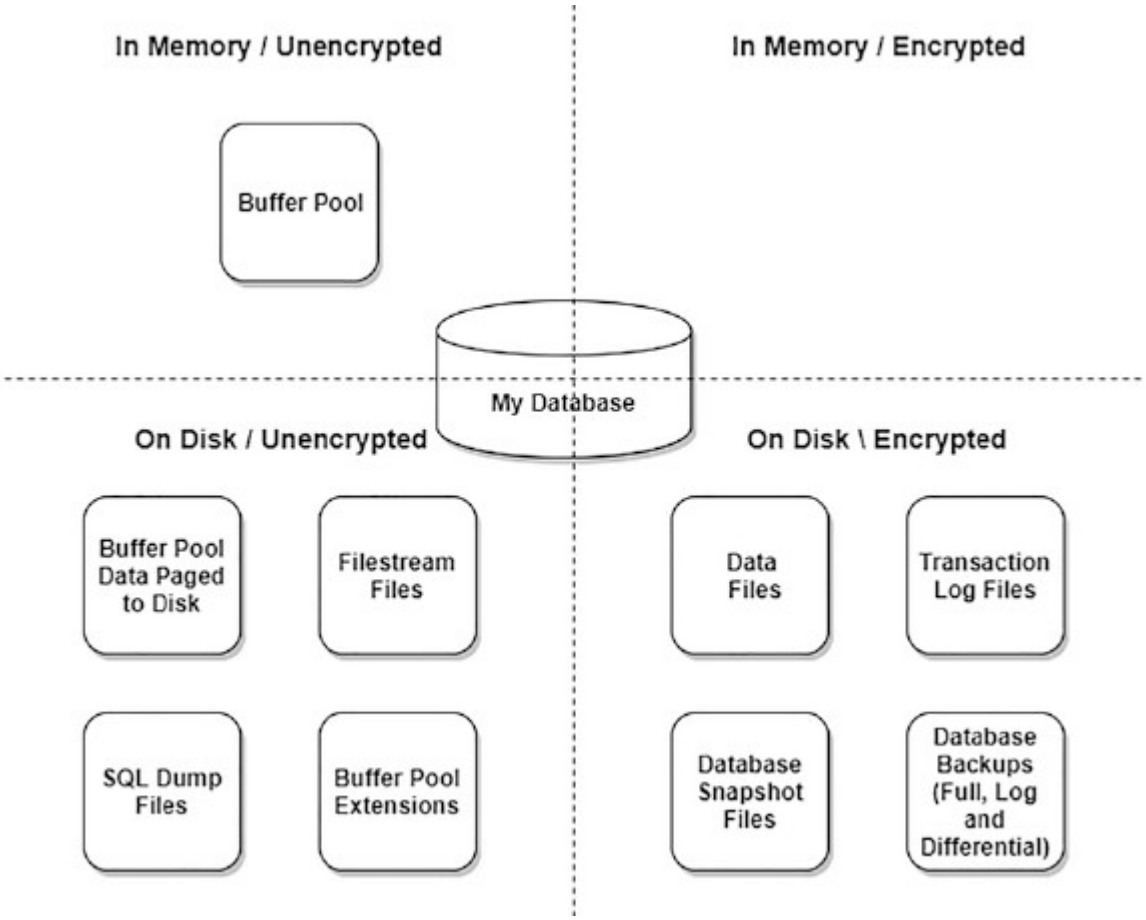
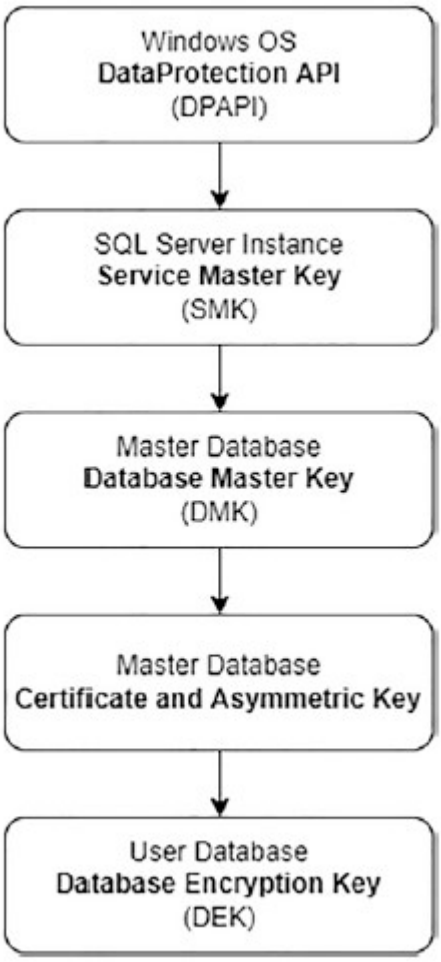
TDE

TDE protects our data stored on disk, what we often refer to as “at-rest” data.

- Data Files
- Log Files
- Backup files
- Database snapshot files
- TempDB is by default protected as soon as TDE is enabled on user DB.
- Does not prevent someone with access to DB from reading the data. Prevents someone from running e.g. grep or any such filesystem level search functionality. Does not encrypt in-memory data or data being passed across the network or user query result set, file stream data or data sitting in BPE or if buffer pool gets paged to disk due to memory pressure or SQL dump files.
- Available on SQL Server 2019 and + (standard and Enterprise). Available only on Enterprise before 2019.


In Memory / Unencrypted
In Memory / Encrypted
Buffer Pool
My Database
On Disk / Unencrypted
On Disk \ Encrypted
Buffer Pool Data Paged to Disk
Filestream Files
Data Files
Transaction Log Files
SQL Dump Files
Buffer Pool Extensions
Database Snapshot Files
Database Backups (Full, Log and Differential)

Understanding the Keys



DPAPI – It is the Root level key. The only key which remains unencrypted and is stored at C:\Windows\System32\Microsoft\Protect\S-1-5-18. It must be secured using appropriate file system permissions.

SMK- The SMK is created when you first install your SQL Server instance and is unique to the instance. The SMK is used to protect your DMK. It is in turn protected by the operating system Data Protection API (DPAPI). When SQL Server creates the **Service Master Key (SMK)**, it **encrypts it using the Windows Data Protection API (DPAPI)** tied to the SQL Server **service account**. Two copies (backups) of SMK are created, and both are encrypted by Data Protection API(DPAPI). One copy is stored in SQL Server and another one in SQL Server registry. The copy inside Registry can be found at "HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Microsoft SQL Server\MSSQL15.MSSQLSERVER\Security" and the copy inside SQL Server can be found using TSQL by querying sys.symmetric_keys DMV.

DMK- The DMK is stored in the Master database and is what's used to encrypt the private key. However, the DMK isn't just used for your database and indeed isn't just used for TDE. It can support several activities. There can be only one DMK for your SQL instance. The DMK is in turn encrypted.

Certificate and Asym Keys – The certificate, which contains a public key which can be used to encrypt data and has a reference to a private key which must be used to decrypt data. The two keys together are known as a public/private key pair and are used to encrypt the DEK using asymmetric encryption.

The public key can be held in plain sight as it can only be used to encrypt data. The private key is required to decrypt data. The private key is in turn encrypted to make sure it is protected.

DEK - This is stored in the database itself and is what is used to encrypt and decrypt data in the database. The DEK is a symmetric key, which means the same key is used to both encrypt and decrypt data. Symmetric key encryption is much quicker than asymmetric key encryption which is why such a key is used in this case. We want the "transparent" encryption activities to have as little overhead as possible while keeping things secure.

The DEK is stored encrypted in the database. Encryption wouldn't be very effective if the key was stored in plain sight for anyone to access and use. As it is stored encrypted, even if someone has your files, there is no way for them to access the unencrypted version of the DEK and use it to read your data.

The encrypted DEK is also stored in any database backups.

Understand the need of so many levels of encryption

- Why DMK cannot be used to "directly" encrypt the DEK?
  - We run into problems however as soon as we think about migrating the DMK. Remember there can only be one DMK per instance: what if the instance we are restoring to already has a DMK and that DMK is already used to protect other objects. We can't just replace it. That is why it is critical that there must be an object between the DMK and the DEK; the DMK cannot be used to encrypt the DEK directly.
- Why do we use "Password" to encrypt Certificate Private key "again" when its already encrypted by the DMK?
  - When we backup the Private key using backup certificate command, the command pulls the unencrypted copy from the master database but we

need to make it safe hence we then need this password which again encrypts it before storing on filesystem, so it remains safe there.

- Now the question may be that when we restore this copy over to new server which will have its own DMK, wont the copy of this private key be different from the current server and wont it cause any issue when we restore this DB over to new server.
    - The answer to this is that the restore command decrypts this private key with the password supplied and then again encrypts this with local DMK and stores in Master database but the key value which used for encryption and decryption is not the encrypted value but the decrypted value. Hence even thought its encrypted by the local DMK, it's still the same key.

How Secure is TDE Encryption?

- TDE uses AES (Advanced Encryption Standard) based algorithm (AES_128/192/256). 128/192/256 specify the length of the key.
- Based on current available computer power, it may take from 1000s to Trillions of years to break 128 key encryption (using only available Brute Force Method)

Setting up TDE

- Create DMK – Why do we use the password when creating DMK when its automatically encrypted using SMK?
    - When we create the DMK, a key is generated and two encrypted copies of that key are created and stored in the master database, one encrypted by the SMK and one by the password specified (both encrypted using the AES_256 algorithm).
    - Having the copy encrypted by the password is necessary where you might need to restore a backup of the master database (including the DMK) to a separate SQL Server instance where the original SMK will not be available.
    - In general, we don't need to do this to recover a TDE enabled database to a separate SQL Server instance if you have backups of the certificate and private key; however, we may use the DMK for purposes other than TDE.
        - Certificates signing SPs
        - SSL Certificates
        - Asymmetric Keys used to Encrypt and Decrypt Digital signatures

        - Used to encrypt the certificates and keys which can be used to secure Service Broker dialogues.
        - Can use used to protect column master keys when using Always Encrypted.
- Backup DMK (not necessary but important)
    - It is recommended to back up the DMK. In most cases this backup is not useful in the context of TDE, but as mentioned above, we may have other objects not related to TDE that depend on the DMK, so it is good practice.
- Create the certificate
    - Self-signed and uses Asym Encryption to encrypt DEK
    - Automatically encrypted by DMK
- Create the DEK
- Encrypt the database

Questions:

- How long can it take for a DB to get encrypted?
    - Do a stress test.
- What type of issues can we face during encryption?
    - Watch out for CPU and IO load
    - Keep an eye on blocking
    - Monitor logs (Logs are not truncated during the process)
    - Keen an eye on sync process if DB is part of AG/LS
- What to do if you run into performance issue due to Encryption?
    - Pause using TF 5004 (if running <2019) or Run ALTER command (if SQL>=2019)
- What if Encryption scan fails?
    - It is possible for the encryption scan to fail. The most likely scenario is a failure due to corruption in your database. Because the scan is a background process you won't get an error message telling you that it has stopped, so it is good to use the methods we've looked at in this chapter to check when the scan is complete.

- Prior to SQL 2019 a failure would exhibit itself by the *encryption_state* column showing as 2 (in progress) in *dm_database_encryption_keys* and the *percent_complete* column showing as zero. Usually, the 5004 trace flag would also have been enabled to prevent the scan from resuming.
- From SQL 2019 we have a little more information and can see in *dm_database_encryption_keys* that *encryption_scan_state_desc* is set to aborted and can tell when that happened.
- In either case your first action should be to check the database for corruption and resolve that if there are any issues. After that you can attempt to restart the encryption scan using the methods we've just looked at. If you have no corruption, and the scan still fails, then you'll need to reach out to Microsoft for support, but that's an unlikely scenario.

- **Taking Backups While Encryption Is in Progress**
  - You can continue to take backups while the scanner is running to encrypt your existing data. However, it is important to understand that until that process is complete your backups will not be fully encrypted.
  - What will happen in the meantime is that a mixture of encrypted and unencrypted data will get written to your backup files. Only backups taken after the TDE encryption scan is complete will be fully protected.

Managing a TDE enabled database

- Migrating or recovering TDE enabled database
  - Create DMK if not exists
  - Restore the certificate and private key from existing server where the original DB exists
- How to recover a DB when you don't have backup of certificate and the private key or you don't have password which was used to backup the private key
  - This will only work if you have the backup of master database from the old server
  - SQL Server was running using a windows service account and you have access to the service account.
- Key rotation
  - Which key should be rotated?
  - How to rotate it?
- TDE performance impact on the server and how to measure it
- Backup Encryption with compression on TDE enabled database.

- Up until the 2016 version, SQL Server did not support backup compression on TDE enabled databases.
- After 2016, backup compression kicks in as soon as your MAXTRANSFERSIZE is greater than 64kb (65536). As long as the value you specify is at least one greater than 64k, then an optimized algorithm for compression of TDE encrypted databases is enabled. Commonly people use the value of 128kb.
- This extra parameter becomes unnecessary if you are on SQL Server 2019 Cumulative Update 5 or higher. With that release, if you specify WITH COMPRESSION for a backup taken for a TDE-protected database and you don't specify MAXTRANSFERSIZE, then MAXTRANSFERSIZE will automatically be increased to 128kb, and your backup will be compressed.

- TDE and high availability
  - Remove the database from HA.
  - Set up TDE for the database and turn on.
  - Set up the keys and certificate on the secondary.
  - Add the database back into HA.