Adatbázis titkosítási módszerek összehasonlítása és folyamatuk leírása különböző adatbázis kezelő rendszerekben. Három DBMS-beli (adatbázis kezelő rendszer) megoldás kerül összehasonlításra, implementálásra és bemutatásra, Microsoft SQL Server (MSSQL), MySQL, PostgreSQL. A következő leírás a hivatalos dokumentációk értelmezése majd alkalmazása alapján készült.

MSSQL

TDE: Alkalmazásához a következő lépéseket kell megtenni.

1. Ún. master key, azaz főkulcs létrehozása
2. Ún. certificate, tanúsítány létrehozása vagy beszerzése, amit ez a főkulcs véd.
3. Adatbázis titkosítási kulcs létrehozása, amit a tanúsítvány véd.
4. Adatbázis beállítása a titkosítás használatára.

Implementálás:

```
----Master adatbázis használata
use master;
go
----Főkulcs létrehozása, amit a megadott jelszóval titkosít.
----Ajánlott bonyolultabb jelszó használata, ez csak szemléltetés célját szolgálja.
create master key encryption by password ='testpw'
go
----Tanúsítvány létrehozása. TDE_CERT a neve, a subject pedig X.509 szabványban
----meghatározott tanúsítvány metaadatainak egy mezőjére utal.
create certificate TDE_CERT with subject ='tde_cert'
go


----Titkosítandó adatbázis kiválasztása
use test2;
go
----Adatbázis titkosítási kulcs létrehozása AES-256 titkosítási algoritmussal,
----amit az előbb létrehozott tanúsítvány véd.
create database encryption key
with algorithm = AES_256
encryption by server certificate TDE_CERT;
go

----A test2 adatbázis beállítása a titkosítás használatára.
go
alter database test2 set encryption on;
go
```

Ez a lépés a Properties > Options > State > Encryption Enabled menüpontban is beállítható.

Minden esetben ajánlott a tanúsítvány és a főkulcs biztonsági mentése is:

```
use master
go
backup certificate TDE_CERT
to file='C:\sql_queries\testkey.cer'
with private key (file='C:\sql_queries\testkey.key', encryption by password ='testpw')
go

backup master key
to file = 'C:\sql_queries\masterkey'
encryption by password = 'testpw';
```

Két módon is meggyőződhetünk a titkosítás sikerességéről. Mielőtt beállítanánk az adatbázist a titkosítás használatára így néz ki annak státusza:

```
select DB.name, DB.is_encrypted, DM.encryption_state from sys.databases DB LEFT OUTER JOIN
sys.dm_database_encryption_keys DM ON DB.database_id = DM.database_id;
```

	name	is_encrypted	encryption_state
1	master	0	NULL
2	tempdb	1	3
3	model	0	NULL
4	msdb	0	NULL
5	test2	0	1

Az ’is_encrypted’ értéke 0 vagy 1 lehet. 0 esetében nem igaz, 1 esetében igaz.

Az ’encryption_state’ több értéket is felvehet, valószínúleg az 1, 2 vagy 3-mal fogunk leginkább találkozni. 1 – titkosítva, 2 – titkosítás folyamatban, 3 – titkosítva.

Beállítás után az értékek:

	name	is_encrypted	encryption_state
1	master	0	NULL
2	tempdb	1	3
3	model	0	NULL
4	msdb	0	NULL
5	test2	1	3

, meggyőződhetünk a titkosítás sikerességéről.

Column Level Encryption

Oszlopszintű titkosításhoz szimmetrikus kulcs használható. A végrehajtáshoz szükség van egy adatbázis főkulcsra, illetve hogy ’createés ’alter’ hívásokat tudjunk végezni az adatbázison.

A folyamat titkosítási része a következőből áll:

Csak egy oszlop került titkosításra, de ahogy a módszer erősségeinél említettem, lehet több oszlopot is kódolni különböző kulcsok és tanúsítványok használatával.

```
----Főkulcs létrehozása, amit a megadott jelszóval titkosít.
----Ajánlott bonyolultabb jelszó használata, ez csak szemléltetés célját szolgálja.
create master key encryption by password ='testpw'
go
----Tanúsítvány létrehozása. COLUMN_CERT a neve, a subject pedig X.509 szabványban
----meghatározott tanúsítvány metaadatainak egy mezőjére utal.
create certificate COLUMN_CERT with subject ='Oldal név és tartalom'
go
----Szimmetrikus kulcs lértehozása, AES-256 titkosítási algoritmussal,
----amit az előbb létrehozott tanúsítvány véd.
create symmetric key Sym_key
    with algorithm = AES_256
    encryption by certificate COLUMN_CERT;
go

----Page táblán belül új oszlop létrehozása titkosított adat tárolására.
use test2;
ALTER TABLE dbo.page
    ADD encryptedPageName varbinary(MAX);
GO

----Szimmetrikus kulcs 'megnyitása', amit az adott tanúsítvány dekódol.
open symmetric key Sym_key decryption by certificate COLUMN_CERT;
go

----Titkosítás ezekben a sorokban történik a megadott szimmetrikus kulccsal.
UPDATE test2.dbo.page
set encryptedPageName = EncryptByKey(Key_GUID('Sym_key'), pageName) from test2.dbo.page;
go

----Kulcs bezárása
close symmetric key Sym_key;
go
```

A dekódolás a következő:

```
----Szimmetrikus kulcs 'megnyitása', amit az adott tanúsítvány dekódol.
open symmetric key Sym_key decryption by certificate COLUMN_CERT;
go

----Adatokat kiválasztom és kiiratom, nem mentem le mégegyszer.
----A dekódolás a DecryptByKey függvény használatával történik,
----majd a bináris eredményt átkonvertálom varchar típusúvá.
SELECT pageName, encryptedPageName
    AS 'Titkosítás után',
    CONVERT(varchar, DecryptByKey(encryptedPageName))
    AS 'Dekódolás után'
    FROM test2.dbo.page;
GO

----Kulcs bezárása
close symmetric key Sym_key;
go
```

A titkosítás és dekódolás eredménye:

	pageName	Titkosítás után	Dekódolás után
1	New Page 0	0x0048EA68F9CE984894B30434F7A0D8F0020000000C14C285...	New Page 0
2	New Page 1	0x0048EA68F9CE984894B30434F7A0D8F00200000000B9191B0...	New Page 1
3	New Page 2	0x0048EA68F9CE984894B30434F7A0D8F0020000000AFE1A83...	New Page 2
4	New Page 3	0x0048EA68F9CE984894B30434F7A0D8F0020000000F39A122...	New Page 3
5	New Page 4	0x0048EA68F9CE984894B30434F7A0D8F00200000000418206F...	New Page 4
6	New Page 5	0x0048EA68F9CE984894B30434F7A0D8F00200000000ED8A2B...	New Page 5
7	New Page 6	0x0048EA68F9CE984894B30434F7A0D8F0020000000ECB8F8...	New Page 6
8	New Page 7	0x0048EA68F9CE984894B30434F7A0D8F0020000005FB47C5...	New Page 7
9	New Page 8	0x0048EA68F9CE984894B30434F7A0D8F0020000000E9E3CB...	New Page 8
10	New Page 9	0x0048EA68F9CE984894B30434F7A0D8F00200000065C865E...	New Page 9
11	New Page 10	0x0048EA68F9CE984894B30434F7A0D8F0020000003D78CD...	New Page 10
12	New Page 11	0x0048EA68F9CE984894B30434F7A0D8F0020000000102C168...	New Page 11
13	New Page 12	0x0048EA68F9CE984894B30434F7A0D8F0020000000B2AC448...	New Page 12
14	New Page 13	0x0048EA68F9CE984894B30434F7A0D8F0020000000370BBB3...	New Page 13
15	New Page 14	0x0048EA68F9CE984894B30434F7A0D8F0020000005D34FD...	New Page 14
16	New Page 15	0x0048EA68F9CE984894B30434F7A0D8F0020000003636AF7...	New Page 15
17	New Page 16	0x0048EA68F9CE984894B30434F7A0D8F00200000040061DB...	New Page 16

Cell / Field Level Encryption

Mező szintű titkosítás folyamata nagyrészt megegyezik az oszlop szintű titkosítással. A titkosítási parancson belül kell specifikációkat megadnunk. Egy ilyen titkosítás például:

```
----Titkosítás ezekben a sorokban történik a megadott szimmetrikus kulccsal.
----Csak a meghatározott mezők kerülnek titkosításra. Ebben az esetben annak a page-nek a neve, aminek id-je = 10.
UPDATE test2.dbo.page
set encryptedPageName = EncryptByKey(Key_GUID('Sym_key'), pageName) from test2.dbo.page where pageId = 10;
go
```

Ezt megelőző kód (főkulcs, szimmetrikus kulcs létrehozása, stb…) megegyezik az oszlop szintű titkosításnál bemutatott kóddal.

	pageName	pageText	pageId	noteId	encryptedPageName
1	New Page 0		0	5	NULL
2	New Page 1		1	5	NULL
3	New Page 2		2	5	NULL
4	New Page 3		3	5	NULL
5	New Page 4		4	5	NULL
6	New Page 5		5	5	NULL
7	New Page 6		6	5	NULL
8	New Page 7		7	5	NULL
9	New Page 8		8	3	NULL
10	New Page 9		9	3	NULL
11	New Page 10		10	3	0x0048EA68F9CE984894B30434F7A0D8F002000000780E34...
12	New Page 11		11	3	NULL
13	New Page 12		12	3	NULL
14	New Page 13		13	3	NULL
15	New Page 14		14	3	NULL
16	New Page 15	<html> <head> </hea...	15	3	NULL
17	New Page 16	<html> <head> </hea...	16	4	NULL

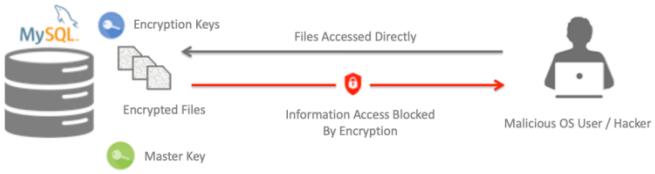
Nyilván, ha több adat felel meg a feltételnek, akkor nem csak egy cella lesz titkosítva. Eredménye ennek a három sornak:

A dekódoláshoz használható az oszlop szintű titkosításnál szemléltetett parancs, nem szükséges a specifikáció. A NULL értékekkel nem fog semmi történni, viszont az id=10-es page neve dekódolásra kerül.

```
----Adatokat kiválasztom és kiiratom, nem mentem le mégegyszer.
----A dekódolás a DecryptByKey függvény használatával történik,
----majd a bináris eredményt átkonvertálom varchar típusúvá.
SELECT pageName, encryptedPageName
    AS 'Titkosítás után',
    CONVERT(varchar, DecryptByKey(encryptedPageName))
    AS 'Dekódolás után'
    FROM test2.dbo.page;
GO
```

	pageName	Titkosítás után	Dekódolás után
1	New Page 0	NULL	NULL
2	New Page 1	NULL	NULL
3	New Page 2	NULL	NULL
4	New Page 3	NULL	NULL
5	New Page 4	NULL	NULL
6	New Page 5	NULL	NULL
7	New Page 6	NULL	NULL
8	New Page 7	NULL	NULL
9	New Page 8	NULL	NULL
10	New Page 9	NULL	NULL
11	New Page 10	0x0048EA68F9CE984894B30434F7A0D8F002000000780E34...	New Page 10
12	New Page 11	NULL	NULL
13	New Page 12	NULL	NULL
14	New Page 13	NULL	NULL
15	New Page 14	NULL	NULL
16	New Page 15	NULL	NULL
17	New Page 16	NULL	NULL

MySQL



Kimondottan a **TDE** alkalmazásáról MySQL-ben csak egy rövid, pár soros leírást találtam, ami a következőket írja le:

„A MySQL TDE lehetővé teszi a nyugvó adatok titksoítását az adatbázis fizikai fájljainak titkosításával. Az adatok titkosítása automatikusan, valós időben, a tárolóba való írás előtt történik, és a tárolóból való olvasáskor dekódolásra kerül. Ennek eredményeképpen a hackerek és a rosszindulatú felhasználók nem tudják az érzékeny adatokat közvetlenül az adatbázisfájlokból kiolvasni.A MySQL TDE a szabványos AES algoritmusokat használja.”

Mivel komplexebb leírást nem találtam, ezért a nyugvó adatokra vonatkozó titkosítási lehetőségeket kellett megnéznem, mivel a TDE is nyugvó adatokat titkosít (data-at-rest). Ezen adatok titkosítását az InnoDB végzi (MySQL tárolási motorja (storage engine)) a következő képpen:

Az InnoDB kétszintű titkosítási kulcsarchitektúrát alkalmaz, amely egy főkulcsból és táblakulcsokból áll.Egy táblatér titkosításakor egy táblatér kulcsot titkosítanak és tárolnak az táblatér fejlécében. Ha egy alkalmazás vagy hitelesített felhasználó szeretne hozzáférni az adatokhoz, az InnoDB a főkulcsot használja a táblatér kulcs dekódolásához.A fő titkosítási kulcs akkor jön létre, amikor a táblatér titkosítása engedélyezve van, és az adatbázison kívül tárolódik.

A nyugalmi adatok titkosítási funkciója egy kulcstartó komponensre vagy bővítményre támaszkodik a titkosítási kulcsok kezeléséhez.

„Amikor a data-at-rest titkosítási funkció központi kulcskezelési megoldást használ, a funkciót "MySQL Enterprise Transparent Data Encryption (TDE)" néven emlegetik.”

A nyugalmi adattitkosítási funkció támogatja az Advanced Encryption Standard (AES) blokkalapú titkosítási algoritmust. A táblatérkulcs titkosításához Electronic Codebook (ECB) blokk titkosítási módot, az adattitkosításhoz pedig Cipher Block Chaining (CBC) blokk titkosítási módot használ.

A ’default_table_encryption’ rendszer változó meghatározza az alapértelmezett titkosítási beállítást sémákra és táblaterekre anélkül, akkor is, ha azok az ’ENCRYPTION’ megadása nélkül lettek volna definiálva.

```
mysql> SET GLOBAL default_table_encryption=ON;
```

Alapértelmezett titkosítási beállítás a séma létrehozásakor vagy módosításakor a ’DEFAULT ENCRYPTION’ megadásával is meghatározható.

```
mysql> CREATE SCHEMA test DEFAULT ENCRYPTION = 'Y';
```

Továbbá titkosítási algoritmusok is használhatóak oszlop és cella adatok titkosítására.

Fontos kikötés titkosított adatok tárolásához, hogy az oszlop adattípusa VARBINARY vagy BLOB típusú legyen, így valószínűleg nem lesznek tárolási problémák. Gyakori titkosítási algoritmusok alkalmazására ad lehetőséget, mint az AES és MD5. Egy pár ilyen funkció megnevezése:

AES_ENCRYPT(), AES_DECRYPT(), MD5(), SHA1(), SHA2(), stb… és ezekhez kapcsolódó tömörítő és dekompressziós funkciók.

Tegyük fel, hogy egy alkalmazás MD5() string értékeket tárol egy CHAR(32) oszlopban:

```
CREATE TABLE md5_tbl (md5_val CHAR(32), ...);
INSERT INTO md5_tbl (md5_val, ...) VALUES(MD5('abcdef'), ...);
```

Egy hex string tömörebb formába történő átalakításához úgy módosítja az alkalmazást, hogy helyette az UNHEX() és a BINARY(16) parancsot használja az alábbiak szerint:

```
CREATE TABLE md5_tbl (md5_val BINARY(16), ...);
INSERT INTO md5_tbl (md5_val, ...) VALUES(UNHEX(MD5('abcdef')), ...);
```

Oracle Database

TDE titkosításhoz a következőkre van szükség:

A fő titkosítási kulcs tárolására szolgáló kulcstároló (keystore). A kulcstároló egy operációs rendszerfájl, amely az adatbázison kívül található. Az adatbázis a kulcstárat használja fel a fő titkosítási kulcs tárolására. A keystore létrehozásához használható az ADMINISTER KEY MANAGEMENT parancs. A keystore titkosítása egy jelszó megadásával történik, amit titkosítási kulcsként használ. A tároló tartalmához csak azok férhetnek hozzá, akik ismerik a jelszót.

Használhatók hardware-es vagy software-es kulcstárolók. Egy software-es kulcstároló egy fájlban kerül meghatározásra, amit az alkalmazó hoz létre egy könyvtárban. Ezek közé tartozik az ún. 'Password-based keystore', 'Auto-login keystore' és 'Auto-login local keystore'. A hardware-es kulcstároló egy hardware-es biztonsági modullal együtt használatos, amely egy olyan fizikai eszköz, amelyet a titkosítási kulcsok biztonságos tárolására terveztek.

Ezt követően, amikor a felhasználó adatokat ad meg, az Oracle Database a következő lépéseket hajtja végre:

-A főkulcsot lekérdezi a kulcstárból.

-A főkulcs segítségével visszafejti a titkosítási kulcsot.

-A titkosítási kulcsot használja a felhasználó által bevitt adatok titkosítására.

-Az adatokat titkosított formában tárolja az adatbázisban.

Ha a felhasználó választja ki az adatokat, a folyamat hasonló: Az Oracle adatbázis visszafejti az adatokat, majd egyszerű szöveges formátumban jeleníti meg azokat.

Adatok konfigurálásának lépési TDE használatához:

Először létre kell hoznia egy kulcstárolót és be kell állítania egy főkulcsot.

Miután létrehozta a kulcstárolót és a fő titkosítási kulcsot, lehetséges az egyes táblaoszlopok vagy egy teljes asztaltér titkosítása.

TDE a következő adattípusokat támogatja: BINARY_FLOAT, BINARY_DOUBLE, CHAR, DATE, BLOB, NCHAR, NUMBER, NVARCHAR2, RAW, TIMESTAMP, VARCHAR2.

Titkosított adatok nem lehetnek idegenkulcsok.

Oszlopok titkosítása grafikusan elvégezhető. Create Table menüpontba a következőket kell elvégezni:

-Titkosítandó oszlop kiválasztása.

-Encryption Options menüpont megnyitása a titkosítási lehetőségek megjelenítésére. Lehetőségek: AES192, 3DES168, AES128, AES256.

-Key Generation menübe random kulcs vagy megadott kulcs generálása. Random kulcs generálása esetén a hash-ekhez kapcsolódó 'Salt' is generálódik.

-Continue, majd elfogadás és kész is.

A DBMS_CRYPTO interfészt biztosít a tárolt adatok titkosításához és visszafejtéséhez, és a hálózati kommunikációt futtató PL/SQL programokkal együtt használható. Támogatást nyújt számosszabványos titkosítási és hash-elési algoritmushoz, beleértve az Advanced Encryption Standard (AES) titkosítási algoritmust is. Ilyen funkciók:

HASH_MD4	128 bites hash-t, vagy message digest-et állít elő a bemeneti üzenetből.
HASH_MD5	Szintén 128 bites hash-t állít elő, de összetettebb, mint az MD4.
HASH_SH1	Biztonságos kivonatoló algoritmus (SHA). 160 bites hash-t állít elő.

ENCRYPT_DES	Adattitkosítási szabvány. Blokk titkosítás. 56 bit hosszúságú kulcsot használ.
ENCRYPT_3DES_2KEY	Adattitkosítási szabvány. Tömbös titkosítás. Egy blokkot 3 alkalommal 2 kulccsal operál. Hatékony kulcshossz 112 bit.
ENCRYPT_3DES	Adattitkosítási szabvány. Blokkos titkosítás. Egy blokkot 3-szor használ.
ENCRYPT_AES128	Fejlett titkosítási szabvány. Blokk titkosítás. 128 bites kulcsméretet használ.
ENCRYPT_AES192	Fejlett titkosítási szabvány. Tömbös titkosítás. 192 bites kulcsméretet használ.
ENCRYPT_AES256	Fejlett titkosítási szabvány. Tömbös titkosítás. 256 bites kulcsméretet használ.
ENCRYPT_RC4	Adatfolyam titkosítás. Titkos, véletlenszerűen generált kulcsot használ, amely minden egyes munkamenethez egyedi.

PostgreSQL

Annak ellenére, hogy egy gyakran használt dbms-ről van szó, a postgresql esetében alig van releváns információ titkosítás szempontjából. Mindössze egy 30-40 soros leírás tartozik a témához, ami az alábbiakat foglalja magába:

PostgreSQL használata több szintű titkosítást tesz elérhetővé.

-Jelszó titkosítás: Az adatbázis-felhasználók jelszavai hash-ként kerülnek tárolásra (a password_encryption beállítással meghatározva), így a rendszergazda sem tudja megállapítani a felhasználóhoz rendelt tényleges jelszót.

-Oszlopszintű titkosítás: A pgcrypto modul lehetővé teszi bizonyos mezők titkosított tárolását. Ez akkor hasznos, ha az adatoknak csak egy része érzékeny. Az ügyfél adja meg a dekódoló kulcsot, az adatokat a szerver dekódolja, majd elküldi az ügyfélnek.

-Adatok titkosítása hálózaton keresztül: Az SSL-kapcsolatok titkosítják a hálózaton keresztül küldött összes adatot: a jelszót, a lekérdezéseket és a visszaküldött adatokat.

-SSL Host hitelesítés: Lehetőség van arra, hogy az ügyfél és a kiszolgáló SSL-tanúsítványokat adjon egymásnak.Megakadályozza, hogy egy számítógép csak annyi ideig adja ki magát kiszolgálónak, hogy el tudja olvasni az ügyfél által küldött jelszót.

-Ügyféloldali titkosítás: Ha a szervergép rendszergazdája nem megbízható, akkor az ügyfélnek titkosítania kell az adatokat, így a titkosítatlan adatok soha nem jelennek meg az adatbázis szolgáltatónak. Az adatok titkosítása a kliensen történik, mielőtt elküldenék őket a szolgáltatónak, és az adatbázis eredményeit a kliensnek kell visszafejteni, mielőtt felhasználná őket.

Ahogy említésre került a pgcrypto modul lehetőséget nyújt kriptográfiai funkciók használatára. Ilyen funkciók:

Általános hash-elési funkció:

- digest(data text, type text) returns bytea – Kiszámítja az adott szöveg bináris hash-ét. A szabványos algoritmusok az md5, sha1, sha224, sha256, sha384 és sha512. Ha a pgcrypto az OpenSSL-lel készült, több algoritmus is rendelkezésre áll.

Kimondottan jelszó hash-elés:

- crypt(password text, salt text) returns text - Kiszámítja a jelszó crypt(3)-stílusú hash-ét. Új jelszó tárolásakor a gen_salt() használatával új ’salt’ értéket kell generálni.

- gen_salt(type text [, iter_count integer]) returns text - Egy új véletlenszerű salt stringet generál a crypt()-ben való használathoz. A salt string azt is megmondja a crypt()-nek, hogy milyen algoritmust használjon.

Összegzés:

Összeszedett, célratörő bemutatást és leírást csak és kizárólag MSSQL-hez találtam. Egyszerűen és gyorsan implementálhatóak voltak a módszerek közvetlen a program telepítése után. Éppen ezért, ha ajánlanom kéne, vagy hosszabb ideig használnom egy dbms-t a felsoroltak közül, biztosan az SQL Server lenne az.

A többi rendszer telepítése után folyamatos hibákba ütköztem a módszerek létrehozásának próbálgatása során. A hibák kezelése sok órát vett igénybe és még így sem minden esetben sikerült megoldani őket, ezért döntöttem úgy, hogy nem kerülnek a módszerek minden dbms esetén implementálásra, inkább csak bemutatásra.

Tekintve, hogy a módszerek nagy része csak bemutatásra került, pontos teljesítmény mérést nem lehet végezni, de mivel gyakran használt dbms-ekről beszélünk, ezért feltételezem, hogy a sikeresen implementált módszerek között nincs olyan nagy teljesítménybeli eltérés.

Valószínűsíthető, hogy kevés rekordot tartalmazó adatbázisok esetén nem lesz hatalmas eltérés a módszerek között, de ha egy adatbázis nagy mennyiségű rekordot tartalmaz, akkor ez a teljesítmény eltérés sokkal jobban érzékelhető lesz.

Úgy gondolom, hogy az MSSQL használata adatmennyiségtől függetlenül mindig egy jó döntés lehet, hiszen az iparágon széleskörbe használt dbms, és titkosítási szempontból egyszerűen implementálhatóak a megoldások.