Лабораторная работа 8

Тема роботи: Шифрування БД

Ціль роботи: дослідження механізмів шифрування БД та їхнього впливу на швидкість операцій читання/запису.

Теоретические сведения

Прозоре шифрування БД (Transparent Data Encryption або TDE) дозволяє шифрувати бази даних повністю. Коли сторінка даних скидається з пам'яті на диск, вона шифрується. Коли сторінка завантажується в оперативну пам'ять, вона розшифровується. Таким чином, база даних на диску виявляється повністю зашифрованою, а в оперативній пам'яті – ні. Основною перевагою TDE є те, що шифрування та розшифровка виконуються абсолютно прозоро для програм. Отже, отримати переваги від використання TDE може будь-яка програма, що використовує для зберігання своїх даних Microsoft SQL Server 2019. При цьому модифікації або доопрацювання програми не потрібно.

На жаль, планується, що Transparent Data Encryption (TDE) буде доступно лише в Enterprise- і Developer-редакціях SQL Server 2019.

ІЄРАРХІЯ КЛЮЧІВ

Зашифрувати дані, як правило, не складає жодних труднощів. Алгоритми шифрування добре відомі і багато з них реалізовані в операційній системі (наприклад, AES). Набагато складніше вигадати, як захистити ключ, яким ці дані зашифровані. Адже якщо ми ”покладемо” його поряд із зашифрованими даними, то ми матимемо в кращому разі обфускацію, але не надійний захист. Для вирішення цього завдання в SQL Server використовується спеціальна ієрархія ключів. У контексті Transparent Data Encryption (TDE) вона будується так:

Для кожної бази даних, яка шифрується за допомогою Transparent Data Encryption (TDE), створюється спеціальний ключ – Database Encryption Key (DEK). Цей ключ використовується для шифрування даних.

Database Encryption Key (DEK) шифрується сертифікатом, який повинен бути створений в БД master.

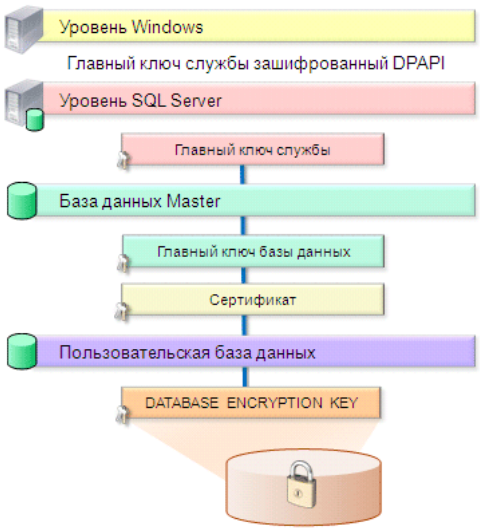
Далі стандартно:

Цей сертифікат шифрується основним ключем БД master.

Головний ключ БД master шифрується головним ключем служби (Service Master Key або SMK).

Головний ключ служби (SMK) шифрується за допомогою DPAPI.

Уся ієрархія наведена на наступному малюнку:



Така схема дозволяє SQL Server у будь-який час отримати доступ до ключа, яким зашифрована БД, а, отже, і до зашифрованих даних. І в той же час, ніхто інший отримати доступ до цих даних не може. Але на жаль, це теорія, а на практиці є дуже обмежений перелік загроз, яким здатне протистояти Transparent Data Encryption (TDE).

TDE

Якщо зловмисник зміг отримати доступ до даних, що захищаються через SQL Server, то Transparent Data Encryption (TDE) виявляється абсолютно марним. Дані зашифровані лише на диску, а пам'яті – ні. Зашифрована база даних виглядає для користувачів так само, як і незашифрована.

Для захисту від адміністраторів Transparent Data Encryption (TDE) також безсило. Адміністратор SQL Server може просто відключити шифрування. Системний адміністратор при бажанні також зможе знайти тисячу та один спосіб отримати доступ до зашифрованих даних (навіть якщо він не є адміністратором SQL Server).

Що реально може зробити Transparent Data Encryption (TDE), то це захистити файли баз даних та резервні копії на випадок їх викрадення. І це вже непогано. Якщо зняти копію з файлів активної БД не так просто (хоча і можливо), то викрадення резервної копії за наявності доступу до них не представляє жодних проблем (які можуть бути проблеми засунути носій з резервною копією в кишеню).

Але тут є свої обмеження. Файли БД та резервні копії будуть надійно захищені, тільки якщо зловмиснику не вдасться разом із даними отримати і ключ. Якщо йому це вдасться, він без проблем розшифрує секретні дані. Найслабшою ланкою тут є ключовий ключ служби (SMK), який знаходиться на вершині ієрархії ключів і який захищається за допомогою DPAPI.

Також слід зазначити, що Transparent Data Encryption (TDE) – це не заміна тим криптографічним можливостям, які є в SQL Server 2019. Якщо шифрування в SQL Server 2005 працює на рівні значень та стовпців (за що часто називається ”cell-level”-шифруванням)), Transparent Data Encryption (TDE) працює на набагато вищому рівні - на рівні бази даних. Завдання, які вирішуються за допомогою обох підходів багато в чому перетинаються, але кожен з них має свої переваги і недоліки. Обидва підходи

використовують одні й самі криптографічні алгоритми (з тими самими довжинами ключів), отже криптографічно обидва підходи забезпечують однаковий рівень захисту даних.

Що саме шифрується?

Коли для БД включено Transparent Data Encryption (TDE), шифруються як її файли даних, і її журнал транзакцій.

Крім того, як тільки на екземплярі SQL Server включається шифрування хоча б однієї бази даних, база даних tempdb також починає шифруватися. За що "постраждала" база даних tempdb, зрозуміло, - вона може містити шматки секретної інформації з баз, що шифруються. А ось за що повинні "страждати" програми, що працюють з іншими, не зашифрованими базами даних? Їхні запити, виконання яких потребує участі бази даних tempdb (великі сортування, наприклад), очевидно, виконуватимуться повільніше. Справа, мабуть, у тому, що не завжди можна визначити джерело даних, які потрапляють у tempdb, і тому для гарантії вона шифрується повністю.

Розберемося по порядку:

Файли даних

Коли для бази даних включається шифрування, SQL Server, як згадувалося вище, в окремому потоці виконує шифрування всіх файлів даних цієї БД. Але є області, що залишаються незашифрованими:

- File Header Page (Page *:0). Це перша сторінка, яка є у всіх файлах даних.
- Boot Page (Page 1:9). Ця сторінка є тільки в першому файлі даних БД і розташована по зміщенню 0x12000 байт від початку файлу. Саме тут зберігається зашифрований сертифікатом Database Encryption Key (DEK), а також майже вся інформація, яка доступна через системне представлення sys.dm_database_encryption_keys. Для відображення вмісту цієї сторінки, крім універсальної команди DBCC PAGE, призначена команда DBCC DBINFO (інформацію, що стосується TDE, вона, на жаль, не повертає).
- Заголовки сторінок (перші 0x60 байт кожної сторінки) також залишаються незашифрованими.

оли SQL Server зашифровує сторінку, він встановлює для неї відповідний прапор (у полі m_flagBits, яке фізично розташоване за зміщенням 4 від початку сторінки і займає 2 байти, встановлюється біт 0x800). Цікаво, що ми ніколи не побачимо цей біт встановленим через DBCC PAGE, тому що у пам'яті всі сторінки розшифровані (хоча у файлі прапорець для цієї сторінки може бути встановлений).

База даних tempdb

Як вже згадувалося раніше, якщо на сервері включається шифрування хоча б для однієї БД, то база даних tempdb теж автоматично починає шифруватися. Причому ситуація з tempdb нагадує ситуацію із журналами транзакцій. Всі нові дані, що записуються в tempdb шифруються, в той час як початкового шифрування даних, які вже є, не виконується. В результаті дані, які вже були tempdb на момент включення шифрування, залишаються незахищеними. Але в даному випадку

вирішити проблему легко, достатньо перезапустити сервер. У цьому випадку tempdb буде створена заново, а всі дані, що записуються в неї, будуть шифруватися з самого початку.

Завдання на лабораторную работу

Завдання 1:

1. Створіть у базі даних симетричний ключ із алгоритмом RSA. Ключ має бути захищений паролем
2. Створіть у базі даних копію таблиці. Всі дані в ній повинні бути зашифровані за допомогою створеного вами симетричного ключа.
3. Виконайте запит, який би повернув усі дані із зашифрованої таблиці
4. Виконати аналогічні операції з використанням асиметричних ключів та сертифікатів.

Завдання 2:

Зашифруйте БД за допомогою прозорого шифрування. Провести аналіз продуктивності та трасування результатів.

Завдання 3:

Використовуючи https://docs.microsoft.com/ru-ru/sql/relational-databases/security/encryption/encrypt-a-column-of-data створіть для таблиці зашифрований стовпець даних.

Завдання 4. Використовуючи https://docs.microsoft.com/ru-ru/sql/relational-databases/security/encryption/always-encrypted-database-engine створити таблицю із зашифрованими за технологією Always Encryption даними, та https://docs.microsoft.com/ru-ru/sql/relational-databases/security/encryption/configure-always-encrypted-using-sql-server-management-studio переглянути зашифровані та розшифровані дані.

Вимога до каталогу звіту:

Звіт містить:

1. Тестові БД
2. Скрипти всіх виконаних операцій шифрування
3. Документ зі скринами результатів:

- про виконання запитів на читання зашифрованих та розшифрованих даних;
- моніторінг продуктивності та трасування процесів.

Хід роботи

Шифрування за допомогою сертифікатів для захисту ключів

Створення сертифіката виконується за допомогою CREATE CERTIFICATE. Найпростіший варіант цієї команди виглядає так:

```
CREATE CERTIFICATE Cert1
ENCRYPTION BY PASSWORD = '11'
```

```
WITH SUBJECT = 'Проверка шифрования',
START_DATE = '02/06/2009'
```

Звернемо увагу, що для створення сертифіката нам не потрібний ніякий центр сертифікації – всі необхідні засоби вже вбудовані в SQL Server.

Параметр ENCRYPTION BY PASSWORD визначає пароль, який буде потрібний для розшифровування даних, захищених сертифікатом (для шифрування даних він не потрібен). Якщо цей параметр пропустити, то сертифікат, що створюється, буде автоматично захищений головним ключем бази даних (Database Master Key). Цей ключ автоматично не створюється. Щоб отримати можливість працювати з ним, потрібно заздалегідь його створити:

```
CREATE MASTER KEY ENCRYPTION BY PASSWORD = 'P@ssw0rd';
```

Крім пароля, головний ключ бази даних автоматично захищається головним ключем служби (Service Master Key). Цей ключ автоматично генерується SQL Server в процесі інсталяції. Треба бути дуже уважним при використанні головного ключа бази даних: якщо ми перевстановимо сервер (а отже, зміниться головний ключ служби), зашифровані дані можуть бути втрачені. Щоб цього не сталося, потрібно проводити регулярне копіювання бази даних master або експортувати головний ключ служби до файлу за допомогою команди BACKUP SERVICE MASTER KEY. Обов'язковий параметр SUBJECT команди CREATE CERTIFICATE визначає мету видачі сертифіката.После того, как сертификат создан, его можно использовать для шифрования данных. Для этой цели применяется специальная функция EncryptByCert:

```
insert into Num_cards

values(1, EncryptByCert(Cert_ID('Cert1'), N'111001') )

insert into Num_cards

values(2, EncryptByCert(Cert_ID('Cert1'), N'111002') )

insert into Num_cards

values(3, EncryptByCert(Cert_ID('Cert1'), N'111003') )
```

Для розшифрування використається команда DecryptByCert. Щоб розшифрувати дані, потрібно використати звичайний запит SELECT:

```
select convert(nvarchar(50), DecryptByCert(Cert_ID('Cert1'), Cred_ID, N'11') )
```

from Num_cards

Шифрування асиметричним ключем

Спочатку необхідно створити асиметричний ключ.

```
CREATE ASYMMETRIC KEY ASymKey1

WITH ALGORITHM = RSA_512

ENCRYPTION BY PASSWORD = '11'
```

Зверніть увагу, що крім пароля, тут потрібно вказати довжину створюваного ключа. У нашому розпорядженні три варіанти: 512, 1024 та 2048 біт.

Після цього за допомогою створеного ключа можна шифрувати дані:

```
insert into Num_cards

values (1, EncryptByAsymKey(AsymKey_ID('ASymKey1'), N'111001'))

insert into Num_cards

values (2, EncryptByAsymKey(AsymKey_ID('ASymKey1'), N'111002'))

insert into Num_cards values

(3, EncryptByAsymKey(AsymKey_ID('ASymKey1'), N'111003'))
```

Ми вносимо в таблицю 3 записи з одним і тим самим асиметричним ключем AsymKey1. В результаті дані в таблиці будуть представлені у вигляді набору символів, що не читається.

Для расшифрования воспользуемся функцией DecryptByAsymKey

```
SELECT Convert(nvarchar(50), DecryptByAsymKey(AsymKey_ID('ASymKey1'),Cred_ID, N'11'))

FROM Num_cards
```

Шифрування симетричним ключем

При використанні симетричних ключів шифрування проводиться швидше, ніж при застосуванні асиметричних алгоритмів, тому при роботі з великими обсягами даних рекомендується використовувати їх. Використання симетричних ключів виглядає дуже схожим. Щоправда, є й невеликі відмінності. По-перше, під час створення симетричного ключа його можна

захищати як паролем, а й іншим симетричним ключем, асиметричним ключем чи сертифікатом. По-друге, під час створення симетричного ключа ви можете вказати один з восьми алгоритмів шифрування, які підтримує SQL Server 2019 DES, Triple DES, TRIPLE_DES_3KEY, RC2, RC4, RC4 з 128-розрядним ключем, DESX, AES зі 128-розрядним ключем, AES зі 192-розрядним ключем та AES з 256-розрядним ключем. (DES, TRIPLE_DES, RC2, RC4, DESX, AES_128, AES_192, AES_256). Саме створення симетричного ключа може мати такий вигляд:

!Примітка!

Починаючи з версії SQL Server 2016 (13.x); всі алгоритми, відмінні від AES_128, AES_192 та AES_256, є нерекомендованими. Щоб використовувати старі алгоритми (не рекомендується), необхідно встановити рівень сумісності бази даних 120 або нижче.

```
CREATE SYMMETRIC KEY SymKey1

WITH ALGORITHM = DES

ENCRYPTION BY PASSWORD = '11'
```

Перед використанням ключа потрібно обов'язково відкрити. Це достатньо зробити лише один раз протягом сеансу роботи користувача:

```
OPEN SYMMETRIC KEY SymKey1 DECRYPTION BY

PASSWORD = '11'
```

Використовуємо створений ключ для шифрування даних:

```
insert into Num_cards

values(1, EncryptByKey(Key_GUID('SymKey1'), convert(nvarchar(50),'111001') ))

insert into Num_cards

values(2, EncryptByKey(Key_GUID('SymKey1'), convert(nvarchar(50),'111002') ))

insert into Num_cards

values(3, EncryptByKey(Key_GUID('SymKey1'),

convert(nvarchar(50),'111003') ))
```

Зверніть увагу, що при розшифровці даних не потрібно передавати функції DecryptByKey ім'я симетричного ключа та пароль. Дані відкритого ключа будуть автоматично підставлені за допомогою команди open. Для розшифровки повідомлення слід запустити наступний скрипт:

```
select convert(nvarchar(50), DecryptByKey(Cred_ID))

from Num_cards
```

Тому як зашифровані дані не можна зберігати в столбцях типу int, char, застосовується команда convert.

Шифрування паролем

SQL Server дозволяє шифрувати дані також просто за допомогою пароля. Для цього використовується функція EncryptByPassPhrase.

У найпростішому варіанті вона приймає лише пароль та дані, які необхідно зашифрувати:

```
insert into Num_cards

values(4, EncryptByPassPhrase('Password', N'111004'))
```

Розшифровка здійснюється за допомогою функції DecryptByPassphrase:

```
select

convert(nvarchar(50), DecryptByPassPhrase('Password', Cred_Id))

from Num_cards
```

Прозоре шифрування

Порядок увімкнення шифрування:

1. Створити головний ключ
2. Створити чи отримати сертифікат, захищений головним ключем
3. Створити ключ шифрування бази даних та захистити його за допомогою сертицикату
4. Задати ведення шифрування бази даних

Перейдемо до самої реалізації:

1) Створення головного ключа шифрування

```
CREATE MASTER KEY ENCRYPTION BY PASSWORD = 'StrongPassword#1';
```

Созданный наш ключ можно увидеть в view:

```
select * from sys.key_encryptions
```

Видалити ключ можна інструкцією:

```
drop master key
```

Після того, як створили головний ключ, необхідно зробити його резервну копію та помістити резервну копію у надійне місце:

```
BACKUP MASTER KEY TO FILE = 'c:\sqltest2012_masterkey_backup.bak'
 ENCRYPTION BY PASSWORD = 'Password1'
```

2) Створення сертифікату

```
CREATE CERTIFICATE TDECertificate WITH SUBJECT ='TDE Certificate for DBClients'
```

Перевірка наявності створеного сертифіката:

```
select * from sys.certificates where name='TDECertificate'
```

Створення резервної копії сертифіката із закритим ключем:

```
BACKUP CERTIFICATE TDECertificate
 TO FILE = 'c:\sqltest2012_cert_TDECertificate'
 WITH PRIVATE KEY
 (
 FILE = 'c:\sqltest2012SQLPrivateKeyFile',
 ENCRYPTION BY PASSWORD = 'Password#3'
 );
```

3) Створення ключа шифрування в нашій базі даних, використовуючи наш сертифікат

```
USE [DBClients]
CREATE DATABASE ENCRYPTION KEY
WITH ALGORITHM = AES_128
ENCRYPTION BY SERVER CERTIFICATE TDECertificate;
```

4) І нарешті включаємо шифрування для нашої бази даних

```
ALTER DATABASE [DBClients]
SET ENCRYPTION ON ;
```

У результаті маємо базу даних із прозорим шифруванням.

Перевірити, що Database Encryption Key (DEK) дійсно створено, можна за допомогою системного представлення sys.dm_database_encryption_keys.

```
SELECT DB_NAME(database_id), * FROM sys.dm_database_encryption_keys
```

	(No column name)	database_id	encryption_state	create_date	regenerate_...	modify_date	set_date	opened_date
1	MySecretDB	9	1	2008-01-14 ...	2008-01-14 ...	2008-01-14...	NULL	2008-01-14 ...

У цей момент все готове для того, щоб увімкнути шифрування бази даних. Вмикаємо.

```
ALTER DATABASE MySecretDB SET ENCRYPTION ON
```

З цього моменту починається процес початкового шифрування бази даних. Він виконується "у фоні" в окремому потоці. Відстежити прогрес виконання цієї операції можна за стовпцем percent_complete вже згаданого раніше системного представлення sys.dm_database_encryption_keys. Так, якщо виконати наведений нижче запит у процесі виконання початкового шифрування бази даних, ми можемо отримати, наприклад, наступний результат

```
SELECT DB_NAME(database_id), encryption_state, percent_complete FROM
sys.dm_database_encryption_keys
```

	(No column name)	encryption_state	percent_complete
1	tempdb	3	0
2	MySecretDB	2	43.20966

А коли процес початкового шифрування бази даних буде завершено, запит поверне наст упний результат:

	(No column name)	encryption_state	percent_complete
1	tempdb	3	0
2	MySecretDB	3	0

У стовпці encryption_state міститься інформація про поточний стан бази даних. Відповідно до SQL Server Books Online (BOL), у контексті Transparent Data Encryption (TDE) БД може бути в одному з наступних станів:

0 - Database Encryption Key (DEK) не створено.
1 - Database Encryption Key (DEK) створено, але база даних не зашифрована.
2 – Виконується початкове шифрування.
3 – База даних зашифрована.

4 - Йде зміна ключа.
5 - Йде розшифровка.

Я думаю, що ви вже звернули увагу, що в результаті включення шифрування для нашої БД база даних tempdb також стала шифруватися. Тому, що саме шифрується для безпеки даних, присвячений наступний розділ.

Журнал транзакцій

На відміну від файлів даних, операція початкового шифрування для журналів транзакцій не виконується. Тобто, інформація про транзакції, яка вже є в журналах транзакцій на момент включення шифрування, залишається незахищеною. Шифрується лише інформація про нові транзакції. Принаймні таку поведінку ми можемо спостерігати у CTP6. Як наслідок, якщо є повна резервна копія БД перед тим, як вона була зашифрована, то навіть після включення шифрування, можна зробити резервну копію журналу транзакцій вже зашифрованої БД, а потім відновити її на момент, що передує шифруванню, і отримати доступ до секретних даних. При цьому відновити архів можна без доступу до ключа (наприклад, на іншому сервері). Наступний сценарій демонструє цю можливість:

```
USE master
go
-- Создаем главный ключ базы данных master
IF(not EXISTS(SELECT * FROM sys.symmetric_keys WHERE name =
'##MS_DatabaseMasterKey##'))
    CREATE MASTER KEY ENCRYPTION BY PASSWORD = 'My$Strong$Password$123'
go
-- Создаем сертификат, которым будем шифровать DEK
CREATE CERTIFICATE DEK_EncCert WITH SUBJECT = 'DEK Encryption Certificate'
go
-- Создаем базу данных, которую будем шифровать
CREATE DATABASE MySecretDB
go
-- И сразу делаем ее полную резервную копию (секретных данных здесь нет)
BACKUP DATABASE MySecretDB TO DISK = 'c:\temp\MySecretDB.bak' WITH INIT
go
USE MySecretDB
go
-- Создаем таблицу и заполняем ее секретными данными
-- Делаем это в транзакции с меткой T1
BEGIN TRAN T1 WITH MARK

CREATE TABLE dbo.MySecretTable (Data varchar(200) not null)

INSERT dbo.MySecretTable (Data) VALUES ('It is my secret')

COMMIT
go
-- Шифруем базу данных
```

```
CREATE DATABASE ENCRYPTION KEY WITH ALGORITHM = AES_256
ENCRYPTION BY SERVER CERTIFICATE DEK_EncCert
go
ALTER DATABASE MySecretDB SET ENCRYPTION ON
go
```

Перевіряємо, що база даних зашифрована:

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

	(No column name)	encryption_state
1	tempdb	3
2	MySecretDB	3

Робимо резервну копію журналу транзакцій (база даних вже зашифрована):

```
BACKUP LOG MySecretDB TO DISK = 'd:\temp\MySecretDB.trn'
```

Стираємо нашу базу даних, а потім сертифікат, яким ми шифрували її Database Encryption Key (DEK). Нам це потрібно для того, щоб емулювати відновлення БД на іншому сервері.

```
USE master
go
DROP DATABASE MySecretDB
go
DROP CERTIFICATE DEK_EncCert
go
```

Тепер спробуємо відновити базу даних. Спочатку повністю:

```
RESTORE DATABASE MySecretDB FROM DISK = 'd:\temp\MySecretDB.bak' WITH
NORECOVERY
RESTORE LOG MySecretDB FROM DISK = 'd:\temp\MySecretDB.trn'
```

Як і слід очікувати, спроба відновлення бази даних закінчилася помилкою. Сертифікат, яким зашифровано Database Encryption Key (DEK), недоступний.

```
Msg 33111, Level 16, State 3, Line 2
Cannot find server certificate with thumbprint
'0x347D263A185EF41D8EB06AE425F7599AD2D0FCC3'.
Msg 3013, Level 16, State 1, Line 2
RESTORE LOG is terminating abnormally.
```

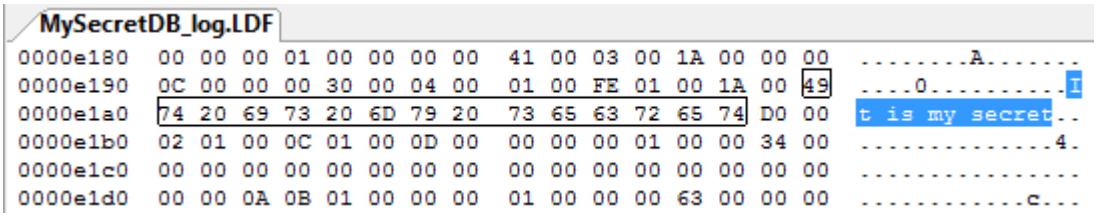
А теперь відновимо базу даних на момент "до включення шифрування", тобто на відмітку T1.

```
RESTORE DATABASE MySecretDB FROM DISK = 'd:\temp\MySecretDB.bak' WITH
NORECOVERY
RESTORE LOG MySecretDB FROM DISK = 'd:\temp\MySecretDB.trn' WITH STOPATMARK
= 'T1'
go
USE MySecretDB
go
-- Запрос к секретным данным
SELECT * FROM dbo.MySecretTable
go
```

Все, доступ к секретним даним мы отримали:

	Data
1	It is my secret

Нагадаю, що самі секретні дані були скинуті в резервну копію журналу транзакцій вже після того, як базу даних було зашифровано та відновлено без жодного доступу до ключа. Насправді, щоб побачити секретні дані, у нашому випадку достатньо було відкрити в редакторі файл журналу транзакцій (або його резервну копію). Незважаючи на те, що наша база зашифрована, секретні дані в журналі транзакцій залишилися у відкритому вигляді (шифруються лише нові транзакції) (перегляд журналу транзакці- https://solutioncenter.apexsql.com/ru/%D0%BF%D1%80%D0%BE%D1%81%D0%BC%D0%BE%D1%82%D1%80-%D1%81%D0%BE%D0%B4%D0%B5%D1%80%D0%B6%D0%B8%D0%BC%D0%BE%D0%B3%D0%BE-ldf-%D1%84%D0%B0%D0%B9%D0%BB%D0%B0/).



Що з цього випливає? Якщо ми шифруємо БД, яка містить секретну інформацію, ми повинні або перестворити журнал транзакцій, або потурбуватися про те, щоб незашифровані транзакції в журналі транзакцій перезаписували нові зашифровані транзакції.