МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждения высшего образования
«Юго-Западный государственный университет»
(ЮЗГУ)

Кафедра информационной безопасности

УТВЕРЖДАЮ
Проректор по учебной работе
_____ О.Г. Локтионова
«_____»_____2017 г.

ШИФРОВАНИЕ SQL SERVER

Методические указания по выполнению лабораторной работы №5
для студентов направления подготовки бакалавриата
10.03.01«Информационная безопасность»

Курск 2017

УДК 621.(076.1)

Составители: А.Г. Спеваков

Рецензент

Кандидат технических наук, доцент кафедры
«Информационная безопасность» И.В. Калуцкий

Шифрование SQL Server [Текст] : методические указания по выполнению лабораторной работы / Юго-Зап. Гос. ун-т; сост.: А.Г. Спеваков. – Курск, 2017. – 48с.: ил. 22, табл. 1. – Библиогр.: с. 48.

Содержат сведения по вопросам проектирования баз данных. Указывается порядок выполнения лабораторной работы, правила содержание отчета.

Методические указания соответствуют требованиям программы, утвержденной учебно-методическим объединением по специальности.

Предназначены для студентов направления подготовки бакалавриата 10.03.01 «Информационная безопасность».

Текст печатается в авторской редакции

Подписано в печать . Формат 60х84 1/16.
Усл.печ. л. 2,79. Уч.-изд. л. 2,53. Тираж 100 экз. Заказ. Бесплатно.
Юго-Западный государственный университет.
305040, г.Курск, ул. 50 лет Октября, 94.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Сообщения о потерях и несанкционированном доступе к конфиденциальным данным появляются все чаще, а значит безопасность баз данных - весьма актуальная проблема для многих компаний. Организации, в чьих базах данных хранится конфиденциальная информация, должны соблюдать требования многочисленных законодательных актов, в том числе Грэмма-Лича-Блайли (GLBA), директивы защиты данных ЕС (EUDPD), акта о преемственности и подотчетности медицинского страхования (HIPAA), стандарта безопасности данных индустрии платежных карт (PCI DSS) и закона Сарбейнса-Оксли (SOX). Эти законы требуют шифрования конфиденциальной информации на уровне базы данных и операционной системы. SQL Server, как и другие распространенные коммерческие системы управления базами данных, располагает множеством вариантов шифрования, в том числе на уровне ячеек, базы данных и файлов через Windows, а также на транспортном уровне.

Эти варианты шифрования обеспечивают безопасность информации на уровне базы данных и операционной системы. Кроме того, они снижают вероятность несанкционированного раскрытия конфиденциальных сведений, даже если поражены инфраструктура или база данных SQL Server. В данной лабораторной работе мы рассмотрим возможности шифрования, реализованные в SQL Server, а также способы шифрования конфиденциальной информации, сохраненной в базах данных SQL Server.

ЦЕЛЬ РАБОТЫ

Изучить теоретическую часть. Рассмотреть основные методы шифрования. Зашифровать базу данных в среде Microsoft SQL Server 2008 R2 любым из представленных способом.

ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

1. Получить задание.
2. Изучить метод шифрования с помощью асимметричного ключа.
3. Создать асимметричный ключ для разработанной БД.
4. Изучить метод управления сертификатами.
5. Создать сертификат для разработанной БД.
6. Изучить метод шифрования с помощью симметричного ключа.
7. Создать симметричные ключи для разработанной БД.
8. Изучить метод прозрачного шифрования.
9. Зашифровать разработанную БД методом прозрачного шифрования.

СОДЕРЖАНИЕ ОТЧЕТА

1. Содержание.
2. Индивидуальное задание.
3. Зашифрованная БД методом асимметричного ключа.
4. Зашифрованная БД методом управления сертификатами.
5. Зашифрованная БД методом симметричного ключа.
6. Зашифрованная БД методом прозрачного шифрования.
7. Контрольные вопросы.

ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

Шифрование - процесс кодирования конфиденциальных данных с использованием ключа или пароля. Шифрование надежно защищает данные и сокращает вероятность несанкционированного раскрытия конфиденциальной информации, так как без соответствующего ключа или пароля данные бесполезны. SQL Server располагает многими режимами шифрования, в том числе на уровне ячеек, базы данных, файлов через Windows и шифрования на транспортном уровне.

Шифрование SQL Server не решает проблему доступности инфраструктуры и баз данных SQL Server, но повышает защищенность данных на уровнях базы данных и операционной системы, даже если нарушена конфиденциальность инфраструктуры или баз данных SQL Server.

Несмотря на то, что шифрование является полезным средством обеспечения безопасности, его не следует применять ко всем данным или соединениям. При решении о внедрении шифрования необходимо проанализировать, как пользователи получают доступ к данным.

Если пользователи получают доступ к данным через открытую сеть, то шифрование может потребоваться для повышения безопасности. Однако если весь доступ осуществляется по безопасной внутренней сети, то шифрование не требуется. Использование шифрования включает политику управления паролями, ключами и сертификатами.

Модель шифрования SQL Server в основном предоставляет функции управления ключами шифрования, соответствующие стандарту ANSI X9.17. В этом стандарте определены несколько уровней ключей шифрования, использующихся для шифрования других ключей, которые в свою очередь применяются для шифрования собственно данных.

В таблице №1 перечислены уровни ключей шифрования SQL Server и ANSI X9.17.

Таблица 1 - уровни ключей шифрования SQL Server и ANSI X9.17

Таблица	Уровень шифрования ключей SQL Server и ANSI X9.17	
Уровень SQL Server	Уровень ANSI X9.17	Описание
SMK	Главный ключ	SMK – ключ верхнего уровня, используемый для шифрования DMK. SMK шифруется с применением DPAPI.
DMK	Ключ шифрования ключей	DMK – симметричный ключ, используемый для шифрования симметричного ключа, ассиметричного ключа и сертификата. Для каждой базы данных может быть определен только один DMK.
Симметричные ключи, ассиметричные ключи и сертификаты	Ключ данных	Симметричные ключи, ассиметричные ключи и сертификаты используются для шифрования данных.

Главный ключ службы Service master key(SMK) - ключ верхнего уровня и предок всех ключей в SQL Server. SMK - асимметричный ключ, шифруемый с использованием Windows Data Protection API (DPAPI). SMK автоматически создается, когда шифруется какой-нибудь объект, и привязан к учетной записи службы SQL Server. SMK используется для шифрования главного ключа базы данных Database master key (DMK).

Второй уровень иерархии ключей шифрования - DMK. С его помощью шифруются симметричные ключи, асимметричные ключи и сертификаты. Каждая база данных располагает лишь одним DMK.

Следующий уровень содержит симметричные ключи, асимметричные ключи и сертификаты. Симметричные ключи -

основное средство шифрования в базе данных. Microsoft рекомендует шифровать данные только с помощью симметричных ключей. Кроме того, в SQL Server 2008 и более новых версиях есть сертификаты уровня сервера и ключи шифрования базы данных для прозрачного шифрования данных.

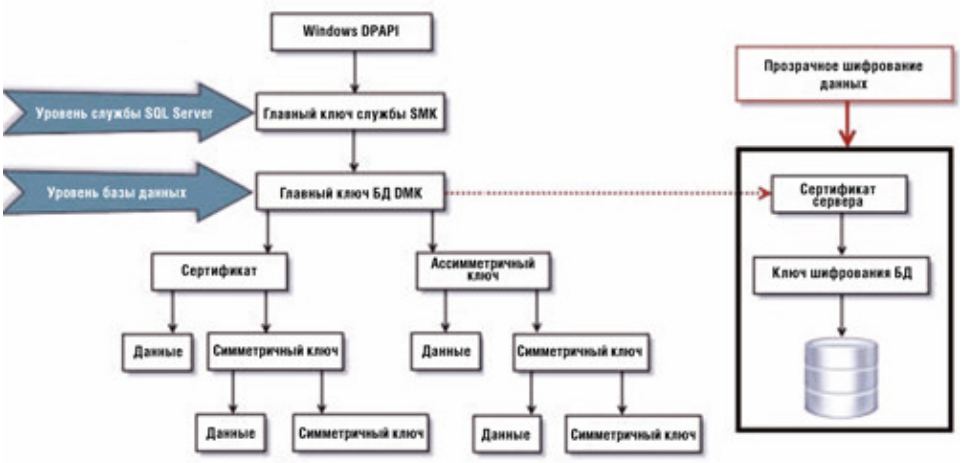

Рис. 1 - Иерархия ключей шифрования для SQL Server 2008

Следует учитывать следующие основные понятия.

- Для лучшей производительности данные следует шифровать с помощью симметричных ключей, а не с помощью сертификатов и асимметричных ключей.
- Главные ключи базы данных защищены главным ключом службы. Главный ключ службы создается при установке SQL Server и шифруется API-интерфейсом защиты данных Windows (DPAPI).
- Возможны другие иерархии шифрования с дополнительными уровнями.
- Модуль расширенного управления ключами хранит симметричные или асимметричные ключи вне SQL Server.
- Прозрачное шифрование данных (TDE) должно использовать симметричный ключ, который называется ключом шифрования базы данных, защищенный сертификатом, который, в свою очередь защищается главным ключом базы данных master или асимметричным ключом, хранящимся в модуле расширенного управления ключами.
- Главный ключ службы и все главные ключи базы данных являются симметричными ключами.

SQL Server поддерживает следующие механизмы шифрования:

- функции Transact-SQL;
- асимметричные ключи;
- симметричные ключи;
- сертификаты
- прозрачное шифрование данных

Функции Transact-SQL - отдельные элементы можно шифровать по мере того, как они вставляются или обновляются, с помощью функций Transact-SQL.

Сертификат открытого ключа, или просто сертификат, представляет собой подписанную цифровой подписью инструкцию, которая связывает значение открытого ключа с идентификатором пользователя, устройства или службы, имеющей соответствующий закрытый ключ. Сертификаты поставляются и подписываются центром сертификации (certification authority, CA). Сущность, получающая сертификат от центра сертификации, является субъектом этого сертификата. Как правило, сертификаты содержат следующие сведения.

- Открытый ключ субъекта.
- Идентификационные данные субъекта, например имя и адрес электронной почты.
- Срок действия, то есть интервал времени, на протяжении которого сертификат будет считаться действительным.

Сертификат действителен только в течение указанного в нем периода, который задается в каждом сертификате при помощи дат (Valid From и Valid To), определяющих начало и окончание срока действия. По истечении срока действия сертификата его субъект должен запросить новый сертификат.

- Идентификационные данные поставщика сертификата.
- Цифровая подпись поставщика.

Эта подпись подтверждает действительность связи между открытым ключом и идентификационными данными субъекта. (В процессе создания цифровой подписи данные, вместе с

некоторыми секретными данными отправителя, преобразуются в тег, называемый подписью.)

Главное преимущество сертификатов в том, что они позволяют не хранить на узлах совокупность паролей отдельных субъектов. Вместо этого узел просто устанавливает доверительные отношения с поставщиком сертификата; после этого поставщик может подписать неограниченное количество сертификатов.

Когда узел (например, защищенный веб-сервер) указывает, что конкретный поставщик сертификата является доверенным корневым центром сертификации, он неявно выражает доверие к политикам, которые поставщик использовал при определении связей для изданных им сертификатов. Таким образом, узел выражает уверенность в том, что поставщик проверил идентификационные данные субъекта сертификата. Узел делает поставщика доверенным корневым центром сертификации, помещая самостоятельно подписанный сертификат поставщика, содержащий открытый ключ поставщика, в хранилище сертификатов доверенного корневого центра сертификации на главном компьютере. Промежуточные или подчиненные центры сертификации являются доверенными только в том случае, если к ним ведет правильный путь от доверенного корневого центра сертификации.

Поставщик может отозвать сертификат до истечения срока его действия. При этом отменяется связь открытого ключа с идентификационными данными, указанными в сертификате. Каждый поставщик ведет список отозванных сертификатов, который можно использовать для проверки конкретного сертификата.

Самостоятельно подписанные сертификаты, созданные SQL Server, соответствуют стандарту X.509 и поддерживают поля X.509 v1.

Асимметричный ключ состоит из закрытого ключа и соответствующего открытого ключа. Каждый из этих ключей позволяет дешифровать данные, зашифрованные другим ключом. На выполнение асимметричных операций шифрования и дешифрования требуется сравнительно много ресурсов, но они

обеспечивают более надежную защиту, чем симметричное шифрование. Асимметричный ключ можно использовать для шифрования симметричного ключа перед его сохранением в базе данных.

Симметричный ключ — это ключ, используемый и для шифрования, и для дешифрования данных. Данные при использовании симметричного ключа шифруются и дешифруются быстро, и он вполне подходит для повседневной защиты конфиденциальных данных, хранящихся в базе данных.

Прозрачное шифрование данных (TDE) является особым случаем шифрования с использованием симметричного ключа. TDE шифрует всю базу данных, используя симметричный ключ, который называется ключом шифрования базы данных. Функция прозрачного шифрования данных (TDE) выполняет в реальном времени шифрование и дешифрование файлов данных и журналов в операциях ввода-вывода. При шифровании используется ключ шифрования базы данных (DEK), который хранится в загрузочной записи базы данных для доступности при восстановлении.

Ключ шифрования базы данных является симметричным ключом, защищенным сертификатом, который хранится в базе данных master на сервере, или асимметричным ключом, защищенным модулем расширенного управления ключами. Функция прозрачного шифрования данных защищает «неактивные» данные, то есть файлы данных и журналов.

Основное правило прозрачного шифрования, это то, что шифрование файла базы данных проводится на уровне страниц. Страницы в зашифрованной базе данных шифруются до записи на диск и дешифруются при чтении в память. Прозрачное шифрование данных не увеличивает размер зашифрованной базы данных. При этом не стоит забывать, что прозрачное шифрование не шифрует данные при передаче через каналы связи, для этого к примеру, можно включить шифрование соединения. Так же надо учитывать, что шифрование\дешифрование данных это нагрузка на CPU, следовательно, при проектировании необходимо учесть запас мощностей процессоров.

При включении функции прозрачного шифрования данных необходимо немедленно создать резервную копию сертификата и закрытого ключа, связанного с этим сертификатом. В случае если сертификат окажется недоступен или понадобится восстановить базу данных на другом сервере либо присоединить ее к другому серверу, необходимо иметь копии сертификата и закрытого ключа. В противном случае будет невозможно открыть базу данных. Сертификат шифрования следует сохранить, даже если функция прозрачного шифрования в базе данных будет отключена. Даже если база данных не зашифрована, части журнала транзакций могут по-прежнему оставаться защищенными, а для некоторых операций будет требоваться сертификат до выполнения полного резервного копирования базы данных. Сертификат, который превысил свой срок хранения, все еще может быть использован для шифрования и расшифровки данных с помощью прозрачного шифрования данных.

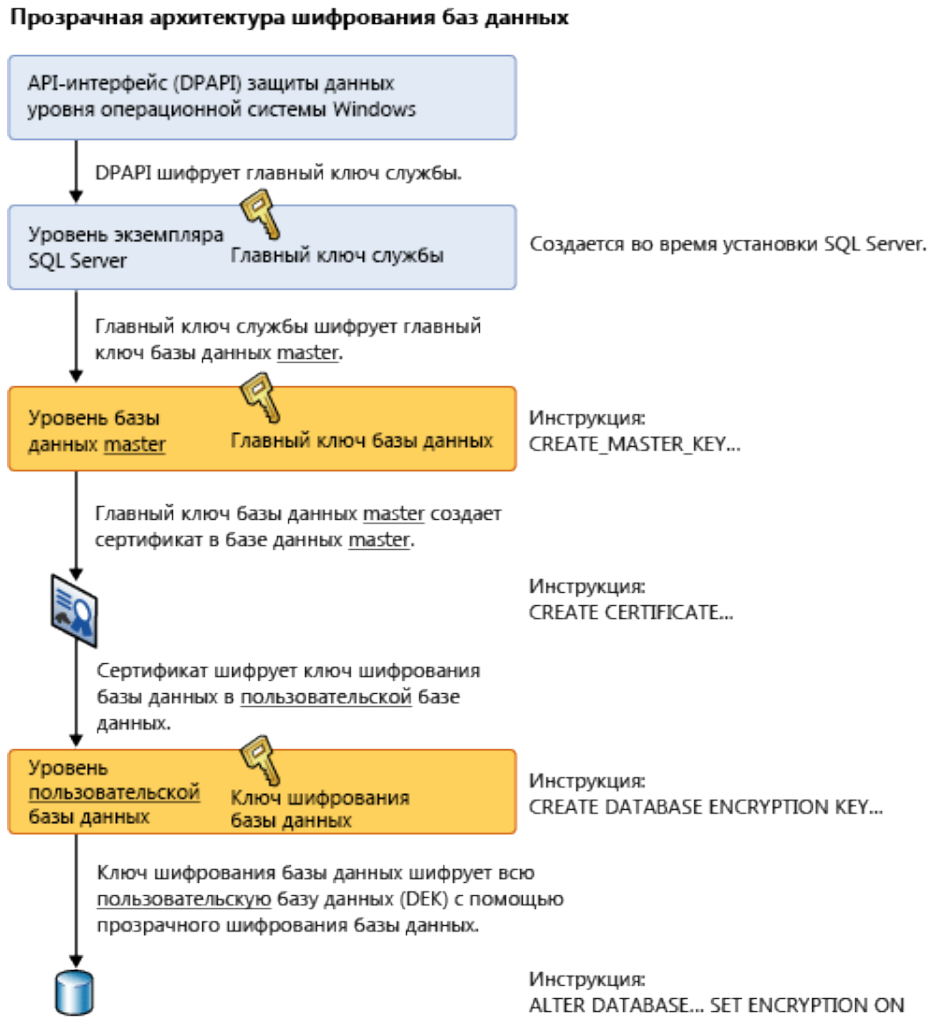

Рис.2 - Архитектура прозрачного шифрования данных

Выбор алгоритма шифрования

Алгоритмы шифрования определяют преобразования данных, исключающие возможность легкого восстановления исходного текста неавторизированными пользователями. SQL Server позволяет администраторам и разработчикам выбирать из нескольких алгоритмов, в том числе DES, Triple DES, TRIPLE_DES_3KEY, RC2, RC4, RC4 с 128-разрядным ключом, DESX, AES с 128-разрядным ключом, AES с 192-разрядным ключом и AES с 256-разрядным ключом.

Не существует одного алгоритма, идеально подходящего для всех случаев. Информация по качеству каждого из них лежит за пределами электронной документации по SQL Server. Однако можно руководствоваться следующими общими принципами:

- Надежные алгоритмы шифрования обычно потребляют больше ресурсов ЦП, чем менее надежные средства шифрования.
- Использование длинных ключей, как правило, дает более надежные результаты, чем шифрование с помощью коротких ключей.
- Асимметричное шифрование слабее симметричного шифрования с использованием ключа той же длины, но является относительно медленным.
- Блочные шифры с длинными ключами надежнее поточных шифров.
- Длинные сложные пароли надежнее, чем короткие пароли.
- При необходимости шифровать большие объемы данных, шифруйте данные с помощью симметричного ключа, а симметричный ключ — с помощью асимметричного ключа.
- Зашифрованные данные не поддаются сжатию, но сжатые данные могут быть зашифрованы. При использовании сжатия данных, выполняйте эту операцию до их шифрования.

Алгоритм RC4 поддерживается только в целях обратной совместимости. Когда база данных имеет уровень совместимости 90 или 100, новые материалы могут шифроваться только с

помощью алгоритмов RC4 или RC4_128. (Не рекомендуется.) Вместо этого используйте более новые алгоритмы, например AES. В SQL Server 2012 и более поздних версиях материалы, зашифрованные с помощью алгоритмов RC4 или RC4_128, могут быть расшифрованы на любом уровне совместимости.

Многократное использование одного и того же KEY_GUID алгоритма RC4 или RC4_128 для разных блоков данных приведет к формированию одинакового ключа RC4, так как SQL Server не предоставляет помеху автоматически. Повторное использование ключа RC4 является типичной ошибкой, снижающей надежность шифра. Таким образом, ключевые слова RC4 и RC4_128 являются устаревшими.

ВЫПОЛНЕНИЕ РАБОТЫ

Рассмотрим шифрование с помощью ассиметричного ключа. Основные функции для управления ассиметричными ключами:

1. CREATE ASYMMETRIC KEY.

Создает асимметричный ключ в базе данных. Асимметричный ключ является защищаемой сущностью на уровне базы данных.

В его форме по умолчанию эта сущность содержит как открытый, так и закрытый ключ. CREATE ASYMMETRIC KEY при выполнении без предложения FROM формирует новую пару ключей. CREATE ASYMMETRIC KEY при выполнении с предложением FROM импортирует пару ключей из файла или открытый ключ из сборки.

По умолчанию закрытый ключ защищается с помощью главного ключа базы данных. Для защиты закрытого ключа необходим пароль, если не был создан главный ключ базы данных. Если главный ключ базы данных существует, пароль необязателен. Закрытый ключ может быть длинной 512, 1024 или 2048 бит.

2. ALTER ASYMMETRIC KEY.

Изменяет свойства асимметричного ключа:

- Изменение пароля закрытого ключа. В следующем примере изменяется пароль, используемый для защиты закрытого ключа асимметричного ключа PacificSales09. Новым паролем будет <enterStrongPasswordHere>.

```
ALTER ASYMMETRIC KEY PacificSales09
WITH PRIVATE KEY (
DECRYPTION BY PASSWORD = '<oldPassword>',
ENCRYPTION BY PASSWORD = '<enterStrongPasswordHere>');
GO
```

- Удаление закрытого ключа из асимметричного ключа. В следующем примере закрытый ключ удаляется из ассиметричного ключа PacificSales19, в котором остается только открытый ключ.

```
ALTER ASYMMETRIC KEY PacificSales19
REMOVE PRIVATE KEY;
GO
```

- Снятие парольной защиты с закрытого ключа. В следующем примере снимается парольная защита закрытого ключа и устанавливается его защита главным ключом базы данных.

```
OPEN MASTER KEY;
ALTER ASYMMETRIC KEY PacificSales09
WITH PRIVATE KEY (
DECRYPTION BY PASSWORD = '<enterStrongPasswordHere>' );
GO
```

3. DROP ASYMMETRIC KEY.

Удаляет асимметричный ключ из текущей базы данных. Нельзя удалять асимметричный ключ, которым в базе данных зашифрован симметричный ключ или с которым сопоставлено имя входа.

Прежде чем удалять такой ключ, следует сначала удалить пользователя или имя входа, с которым этот ключ сопоставлен, либо удалить или перешифровать симметричный ключ, который зашифрован данным асимметричным ключом. Для удаления шифрования, выполненного асимметричным ключом, предназначен параметр DROP ENCRYPTION инструкции ALTER SYMMETRIC KEY.

Метаданные асимметричных ключей доступны через представление каталога sys.asymmetric_keys. Сами ключи для просмотра из базы данных недоступны.

Если асимметричный ключ сопоставлен ключу поставщика расширенного управления ключами на устройстве поставщика, а параметр REMOVE PROVIDER KEY не указан, то ключ будет удален из базы данных, но не с устройства. Будет выдано предупреждение.

4. ENCRYPTBYASYMKEY.

Шифрует данные при помощи асимметричного ключа. Шифрование/дешифрование с помощью асимметричного ключа является очень дорогостоящим по сравнению с шифрованием/дешифрованием с помощью симметричного ключа. Рекомендуется не шифровать асимметричным ключом большие наборы данных, например таблицы с пользовательскими данными. Вместо этого данные следует шифровать с помощью сильного симметричного ключа, а симметричный ключ шифровать с помощью асимметричного.

EncryptByAsymKey возвращает значение NULL, если ввод превышает определенное число байтов, в зависимости от алгоритма. Ограничения: ключ RSA на 512 бит может шифровать до 53 байт, ключ на 1024 бита может шифровать до 117 байт, ключ на 2048 бит может шифровать до 245 байт. (Обратите внимание, что в SQL Server и сертификаты, и асимметричные ключи служат оболочками для ключей RSA.)

5. DECRYPTBYASYMKEY.

Дешифрует данные асимметричным ключом. Шифрование и дешифрование асимметричным ключом являются очень дорогостоящими по сравнению с шифрованием и дешифрованием симметричным ключом. Рекомендуется использовать асимметричный ключ при работе с большими наборами данных, например, с данными пользователей в таблицах.

Переходим непосредственно к шифрованию нашей базы данных. Рассмотрим пример шифрования с помощью асимметричного ключа:

1. Войдём в нашу базу данных (“Библиотека”) и создадим для неё новый запрос. Чтобы создать асимметричный ключ, напишем следующий код:

```
CREATE ASYMMETRIC KEY SampleAsymKey
WITH ALGORITHM = RSA_2048
ENCRYPTION BY PASSWORD = '<Password>'
```

Если всё введено верно вы увидите сообщение об успешном выполнении команд. Во вкладке Безопасность => Асимметричные

ключи нашей базы данных появится созданный нами ассиметричный ключ (Рис 3).

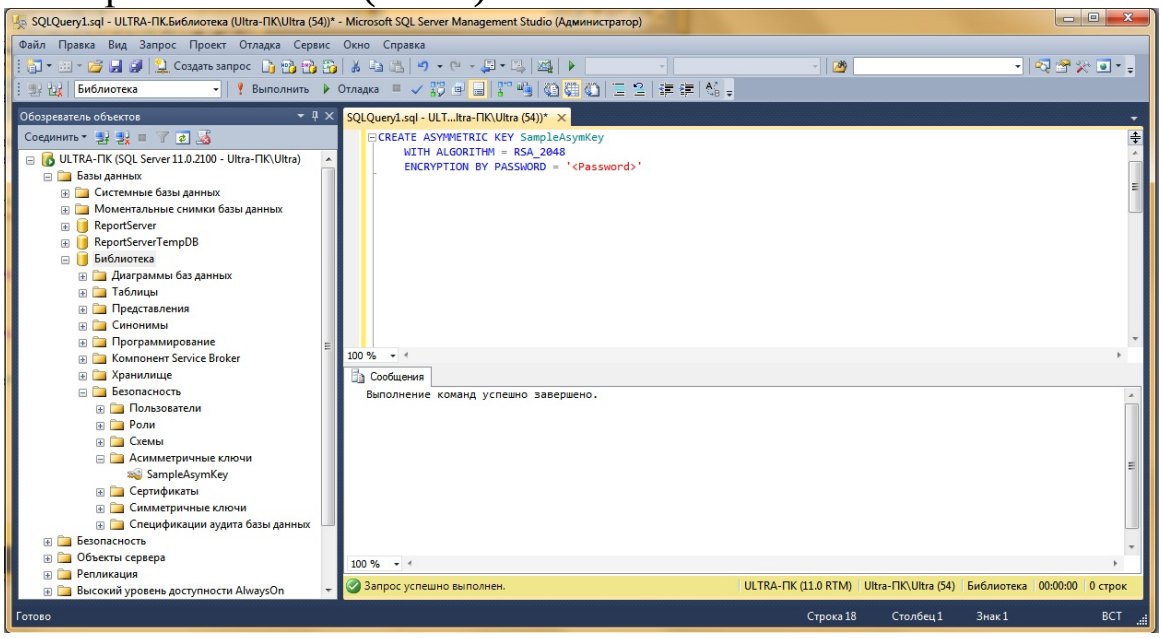

Рис. 3 – Создание асимметричного ключа

2. Следующий шаг – объявление переменных. Объявим две переменные (текстовая и бинарная). В текстовой записываем некий текст и выведем его на экран. Зашифруем его с помощью функции ENCRYPTBYASYMKEY и расшифруем с помощью функции DECRYPTBYASYMKEY (Рис. 4):

```
declare @plaintext nvarchar(60)
declare @ciphertext varbinary(256)

--Инициализировать открытый текст
set @plaintext='Это простой текст'
print @plaintext

--Шифруем
set @ciphertext=ENCRYPTBYASYMKEY
(AsymKey_ID('SampleAsymKey'),@plaintext)
print @ciphertext

--Расшифровка
set @ciphertext=DECRYPTBYASYMKEY
```

(AsymKey_ID('SampleAsymKey'),@ciphertext,N'<Password>')
print cast (@plaintext as nvarchar(max))

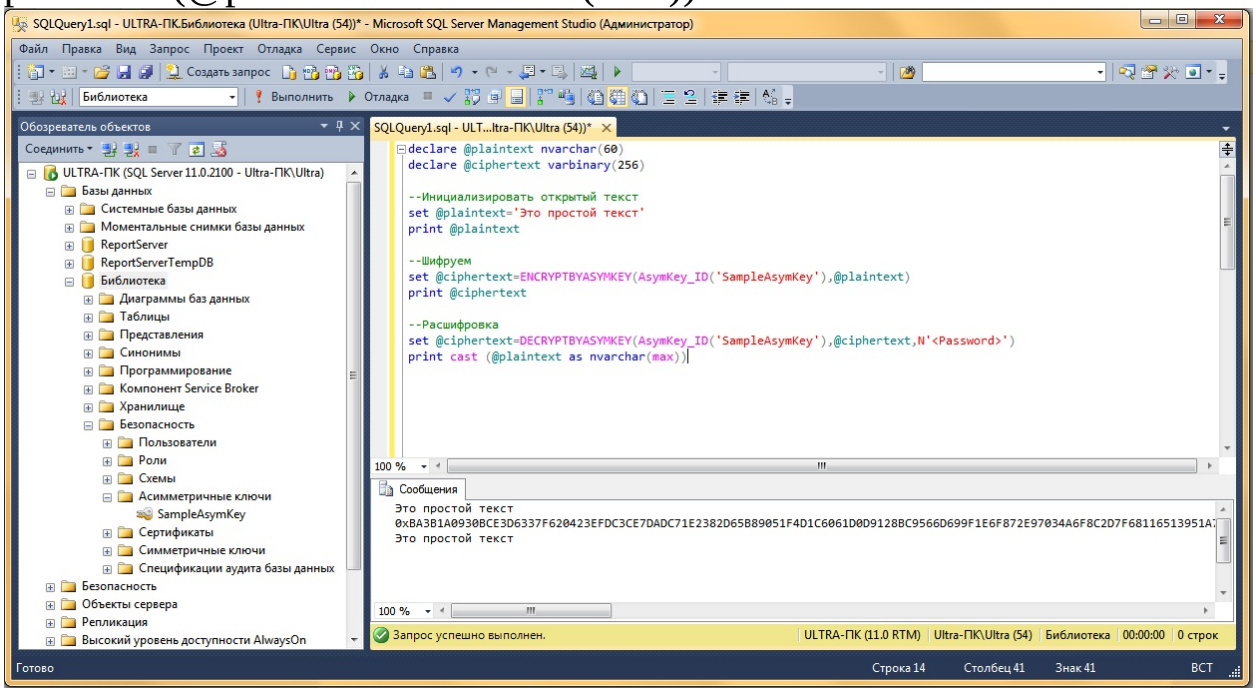

Рис. 4 – Шифрование и дешифрование текста

3. Чтобы удалить асимметричный ключ введите следующий запрос (Рис. 5):

DROP ASYMMETRIC KEY SampleAsymKey

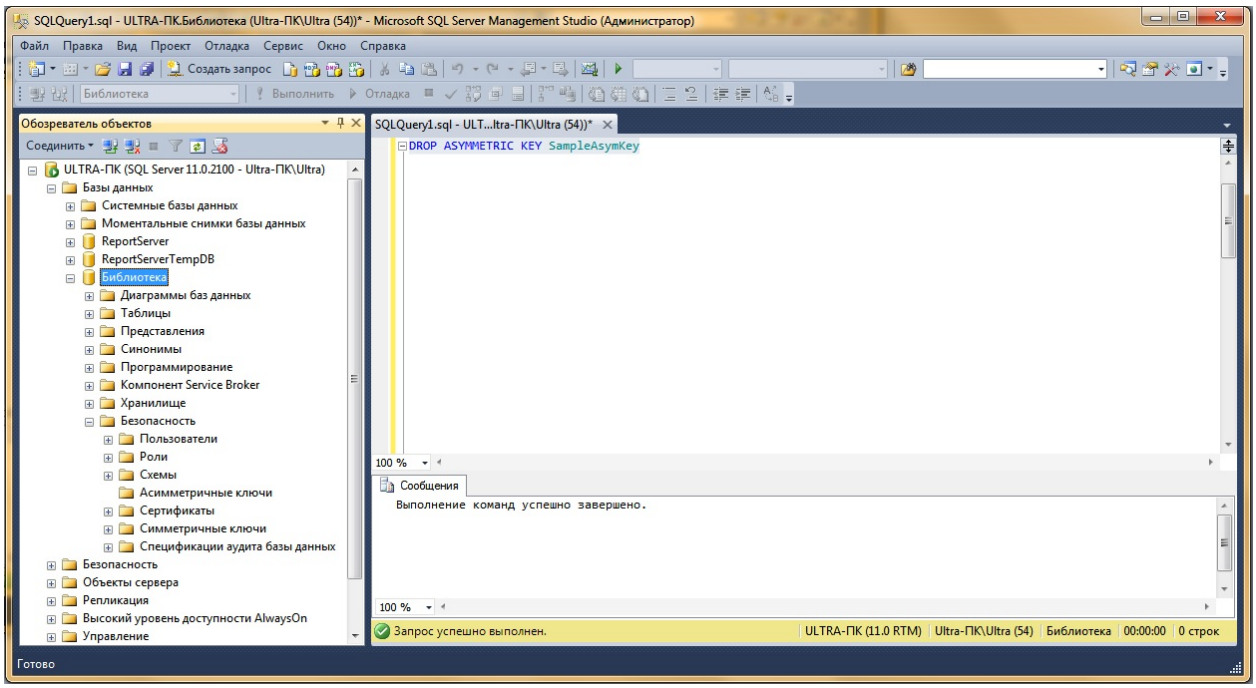

Рис. 5 – Удаление асимметричного ключа

Рассмотрим шифрование с помощью сертификата. Функции для управления сертификатами:

1. CREATE CERTIFICATE - добавляет сертификат в базу данных SQL Server. Сертификат — это защищаемый объект уровня базы данных, соответствующий стандарту X.509 и поддерживающий поля X.509 V1. Инструкция CREATE CERTIFICATE может загрузить сертификат из файла или сборки. Она также может создать пару ключей и самостоятельно подписанный сертификат.

Закрытые ключи, создаваемые SQL Server имеют в длину 1024 бит. Закрытые ключи, импортированные из внешнего источника, имеют минимальную длину в 384 бит и максимальную длину в 4096 бит. Длина импортируемого закрытого ключа должна быть кратной 64 бит. Закрытый ключ должен соответствовать открытому ключу, указанному в аргументе certificate_name.

При создании сертификата из контейнера загрузка закрытого ключа необязательна. Однако в случае, когда SQL Server создает самостоятельно подписанный сертификат, закрытый ключ создается всегда. По умолчанию закрытый ключ шифруется главным ключом базы данных. Если главного ключа базы данных не существует, а пароль не указан, произойдет ошибка при выполнении инструкции.

Если закрытый ключ зашифрован главным ключом базы данных, указывать пароль расшифровки не требуется.

2. ALTER CERTIFICATE - изменяет закрытый ключ, используемый для шифрования сертификата, или добавляет закрытый ключ, если он не существует. Изменяет доступность сертификата для компонента Компонент Service Broker. Закрытый ключ должен соответствовать открытому ключу, заданному в аргументе certificate_name. Предложение DECRYPTION BY PASSWORD может быть опущено, если пароль в файле защищен паролем, равным NULL.

Если закрытый ключ сертификата, уже существующий в базе данных, импортируется из файла, закрытый ключ будет автоматически защищен главным ключом базы данных. Чтобы

защитить закрытый ключ паролем, используйте предложение ENCRYPTION BY PASSWORD.

Параметр REMOVE PRIVATE KEY удалит закрытый ключ сертификата из базы данных. Это можно сделать, если сертификат будет использоваться для проверки подписей или в сценариях компонента Компонент Service Broker, не требующих закрытого ключа. Не удаляйте закрытый ключ сертификата, который защищает симметричный ключ.

Указывать пароль расшифровки не нужно, если закрытый ключ зашифрован с помощью главного ключа базы данных.

- Изменение пароля сертификата:

```
ALTER CERTIFICATE Shipping04
WITH PRIVATE KEY
(DECRYPTION BY PASSWORD = 'pGF$5DGvbd2439587y',
ENCRYPTION BY PASSWORD = '4-329578thlkajdshglXCSgf');
GO
```

- Изменение пароля, используемого для шифрования закрытого ключа:

```
ALTER CERTIFICATE Shipping11
WITH PRIVATE KEY
(ENCRYPTION BY PASSWORD = '34958tosdgfkh##38',
DECRYPTION BY PASSWORD = '95hkjdskghFDGGG4%');
GO
```

- Изменение защиты закрытого ключа паролем на защиту главным ключом базы данных:

```
ALTER CERTIFICATE Shipping15
WITH PRIVATE KEY
(DECRYPTION BY PASSWORD = '95hk000eEnvjkjy#F%');
GO
```

3. DROP CERTIFICATE - удаляет сертификат из базы данных. Резервная копия сертификата, используемого для шифрования базы данных, должна быть сохранена, даже если она больше не включена в базе данных. Даже если база данных больше не зашифрована, части журнала транзакций могут по-прежнему оставаться защищенными, а для некоторых операций будет

требоваться сертификат до выполнения полного резервного копирования базы данных.

Сертификат также потребуется для восстановления из резервных копий, созданных в то время, когда база данных была зашифрована. Сертификаты можно удалять только при условии, что с ними не связано никаких сущностей.

4. BACKUP CERTIFICATE - экспортирует сертификат в файл. Если закрытый ключ зашифрован с паролем в базе данных, необходимо указать пароль для дешифрования.

При создании резервной копии закрытого ключа в файле шифрование является необходимым. Пароль, используемый для защиты резервной копии сертификата, не является тем же ключом, который применялся для шифрования закрытого ключа сертификата.

Чтобы восстановить сертификат из резервной копии, используйте инструкцию CREATE CERTIFICATE.

5. ENCRYPTBYCERT - шифрует данные на открытом ключе сертификата.

Эта функция шифрует данные при помощи открытого ключа сертификата. Код может быть расшифрован только с помощью соответствующего закрытого ключа. Ассиметричные преобразования гораздо более накладны, чем шифрование и дешифрование с использованием симметричного ключа. Поэтому использование ассиметричного шифрования не рекомендуется при работе с большими объемами данных, например, таблицами пользовательских данных.

6. DECRYPTBYCERT - расшифровывает данные при помощи закрытого ключа сертификата.

Эта функция расшифровывает данные при помощи закрытого ключа сертификата. Криптографические преобразования с использованием асимметричных ключей требуют значительных ресурсов. Поэтому функции EncryptByCert и DecryptByCert не подходят для операций шифрования пользовательских данных.

Рассмотрим пример использования функций для управления сертификатами:

1. Вначале создадим папку (Cerf), где будут храниться сертификаты (Рис. 6)

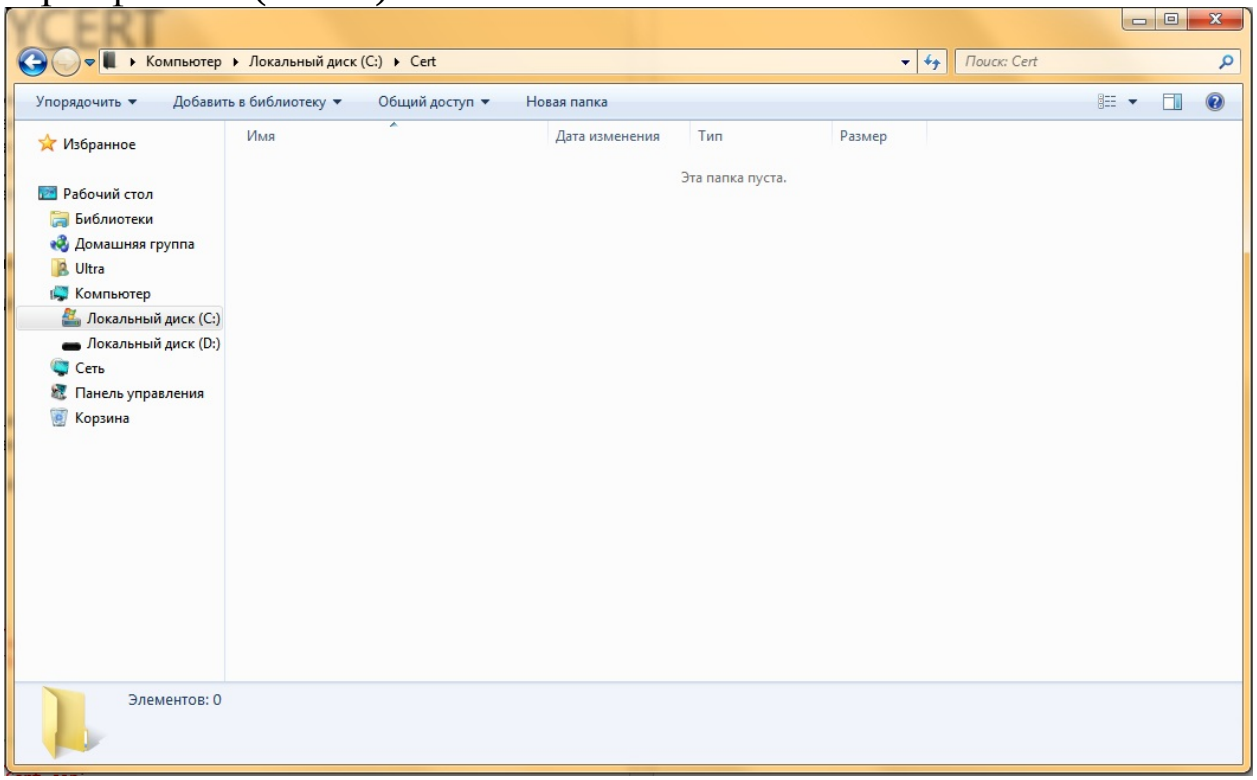

Рис. 6 – Папка для хранения сертификатов

2. Создаём сертификат в нашей базе данных, который шифруется с помощью пароля, указываем тему сертификата и время действия сертификата. Экспортируем его в файловую систему и указываем приватный ключ. Задаём пароль для шифрации и дешифрации (Рис.7):

```
USE Библиотека go
--Создаём сертификат
CREATE CERTIFICATE SampleCert
ENCRYPTION BY PASSWORD = 'Password'
WITH SUBJECT = 'Sample certificate',
EXPIRY_DATE = '2026-10-10';

--Сохраняем сертификат в виде файла
BACKUP CERTIFICATE SampleCert
TO FILE='C:\Cert\BackupSampleCert.cer'
```

WITH PRIVATE KEY (FILE='C:\Cert\BackupSampleCert.pvk',
ENCRYPTION BY PASSWORD='p@ssword',
DECRYPTION BY PASSWORD='Password')

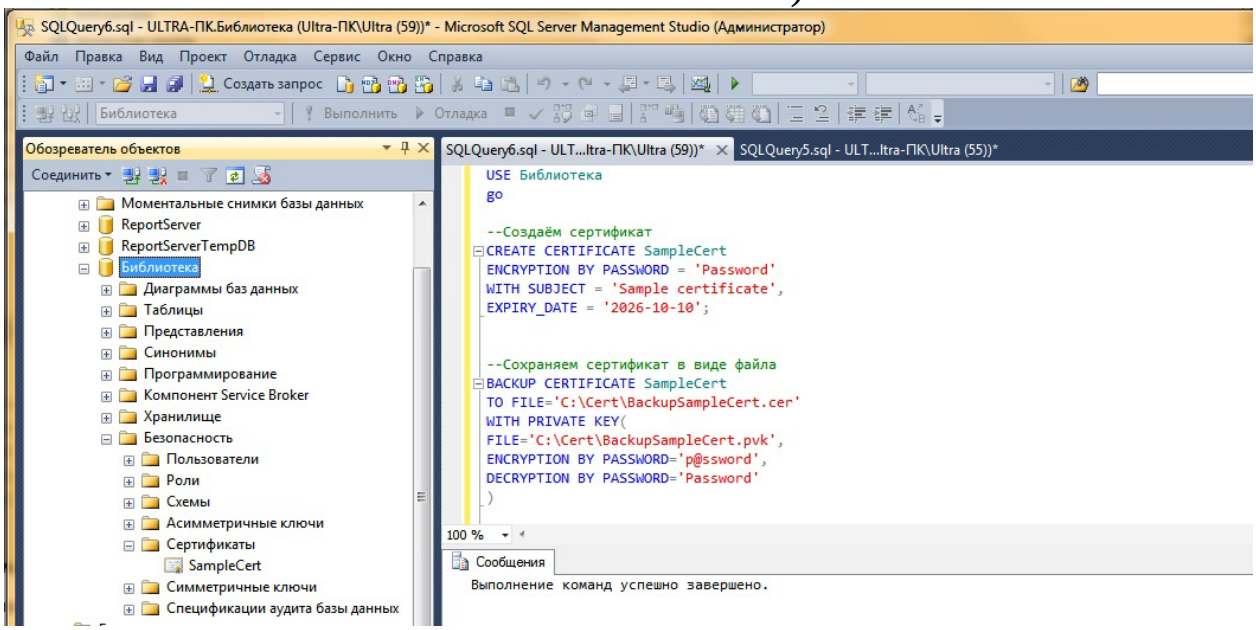

Рис. 7 – Создание сертификата

4. Чтобы удалить сертификат из sql server введите следующий запрос (Рис. 8):

DROP CERTIFICATE SampleCert

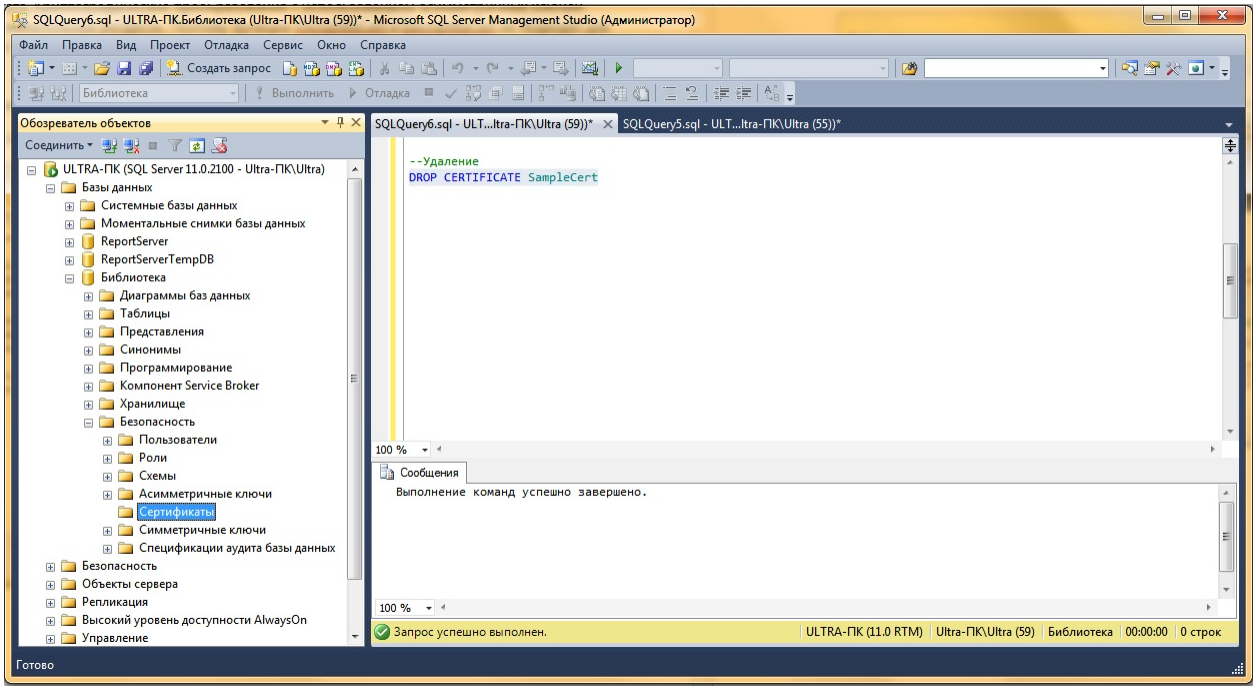

Рис. 8 – Удаление сертификата

5. Чтобы воссоздать из файла введите следующий вопрос (Рис. 9):

```
CREATE CERTIFICATE SampleCert
FROM FILE='C:\Cert\BackupSampleCert.cer'
WITH PRIVATE KEY(
FILE='C:\Cert\BackupSampleCert.pvk',
ENCRYPTION BY PASSWORD='p@ssword',
DECRYPTION BY PASSWORD='Password')
```

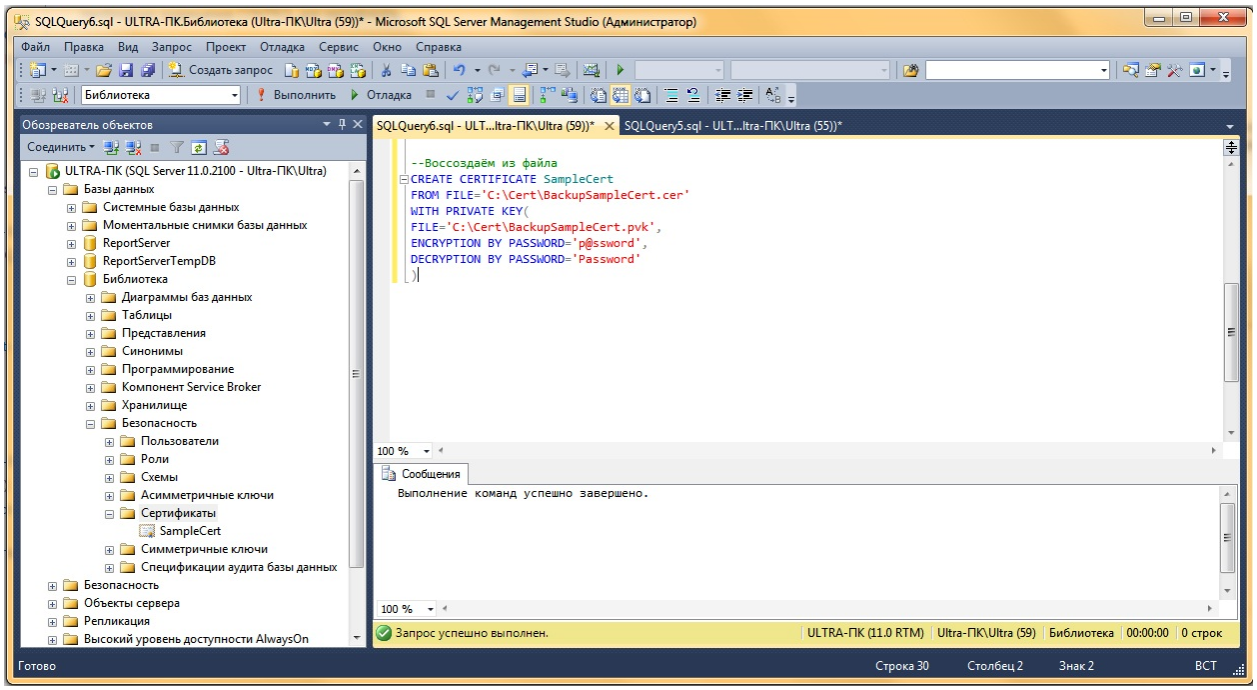

Рис. 9 – Воссоздаём сертификат из созданного файла

Рассмотрим симметричные ключи. Функции для управления симметричными ключами:

1. CREATE SYMMETRIC KEY - создает симметричный ключ и указывает его свойства в SQL Server. После создания симметричный ключ должен быть зашифрован с помощью по крайней мере одного из следующих средств: сертификат, пароль, симметричный ключ, асимметричный ключ или PROVIDER. Ключ может быть зашифрован более чем один раз для каждого типа шифрования. Другими словами, один симметричный ключ может

быть зашифрован с использованием нескольких сертификатов, симметричных ключей и асимметричных ключей одновременно.

Если симметричный ключ шифруется с использованием пароля вместо открытого ключа главного ключа базы данных, то используется алгоритм шифрования TRIPLE DES. Поэтому ключи, созданные с помощью сильных алгоритмов шифрования, таких как AES, защищены с помощью более слабого алгоритма.

Необязательный пароль можно использовать для шифрования симметричного ключа перед распространением ключа нескольким пользователям.

Владельцем временных ключей является пользователь, который создает их. Временные ключи действительны только в текущем сеансе.

Аргумент IDENTITY_VALUE формирует идентификатор GUID, с помощью которого маркируются данные, зашифрованные новым симметричным ключом. Маркирование может быть использовано для сопоставления ключей шифрованным данным.Идентификатор GUID, формируемый указанной фразой, будет всегда одним и тем же. Фраза, использованная для создания идентификатора GUID, не может быть повторно использована до тех пор, пока активен хотя бы один сеанс, использующий эту фразу.Предложение IDENTITY_VALUE является необязательным, но рекомендуется использовать его для хранения данных, зашифрованных с применением временного ключа.

Не существует алгоритма шифрования по умолчанию.

Защищать конфиденциальные данные с помощью потоковых шифров RC4 и RC4_128 не рекомендуется

Сведения о симметричных ключах доступны в представлении каталога sys.symmetric_keys.

Симметричные ключи не могут быть зашифрованы с помощью симметричных ключей, созданных поставщиком шифрования.

Пояснение к алгоритмам DES:

- DESX был именован неправильно. Симметричные ключи, созданные с параметром ALGORITHM = DESX, в действительности используют шифр TRIPLE DES с 192-битным ключом. Алгоритм DESX не предоставляется. В

будущей версии Microsoft SQL Server этот компонент будет удален. Избегайте использования этого компонента в новых разработках и запланируйте изменение существующих приложений, в которых он применяется.

- Симметричные ключи, созданные с параметром ALGORITHM = TRIPLE_DES_3KEY, используют шифр TRIPLE DES с 192-битным ключом.
- Симметричные ключи, созданные с параметром ALGORITHM = TRIPLE_DES, используют шифр TRIPLE DES с 128-битным ключом.

Многократное использование одного и того же RC4 или RC4_128 KEY_GUID для различных блоков данных приведет к одному и тому же ключу RC4, так как SQL Server не предоставляет рассеивание автоматически. Повторное использование одного и того же ключа RC4 является типичной ошибкой, становящейся причиной очень слабого шифрования. Таким образом, ключевые слова RC4 и RC4_128 являются устаревшими. В будущей версии Microsoft SQL Server этот компонент будет удален. Не используйте его при работе над новыми приложениями и как можно быстрее измените приложения, в которых он в настоящее время используется.

Алгоритм RC4 поддерживается только в целях обратной совместимости. Когда база данных имеет уровень совместимости 90 или 100, новые материалы могут шифроваться только с помощью алгоритмов RC4 или RC4_128. (Не рекомендуется.) Используйте вместо этого более новые алгоритмы, например AES. В SQL Server 2014 материалы, зашифрованные с помощью алгоритмов RC4 или RC4_128, могут быть расшифрованы на любом уровне совместимости.

2. ALTER SYMMETRIC KEY - изменяет свойства симметричного ключа. Для изменения шифрования симметричного ключа используйте предложения ADD ENCRYPTION и DROP ENCRYPTION. Невозможно, чтобы ключу не соответствовал никакой способ шифрования. Поэтому оптимальным способом изменения метода шифрования является добавление новой формы шифрования и затем удаление старой.

Для изменения владельца симметричного ключа выполните инструкцию ALTER AUTHORIZATION.

3. DROP SYMMETRIC KEY - удаляет симметричный ключ из текущей базы данных. Если ключ является открытым в текущем сеансе, то инструкция завершится с ошибками.

Если асимметричный ключ был сопоставлен с ключом расширенного управления ключами на устройстве расширенного управления ключами, а параметр REMOVE PROVIDER KEY не был указан, то ключ будет удален из базы данных, но не с устройства; также будет выдано предупреждение.

4. OPEN SYMMETRIC KEY - расшифровывает симметричный ключ и делает его доступным для использования. Открытые симметричные ключи привязаны к сеансу, а не к контексту безопасности. Открытый ключ останется доступным, пока не будет явно закрыт или сеанс не будет прерван. Если открыт симметричный ключ, после чего произошло переключение контекста, ключ останется открытым и будет доступным в олицетворенном контексте.

Если симметричный ключ был зашифрован другим ключом, сначала необходимо открыть этот ключ. Если симметричный ключ уже открыт, то запрос является запросом NO_OP. Если пароль, сертификат или ключ для расшифровки симметричного ключа неверен, запрос завершается сбоем.

Симметричные ключи, созданные из поставщиков шифрования, не могут быть открыты. Операции шифрования и расшифровки, для которых используется симметричный ключ этого типа, выполняются успешно без инструкции OPEN, поскольку открытие и закрытие ключа осуществляется поставщиком шифрования.

5. CLOSE SYMMETRIC KEY - закрывает симметричный ключ или все симметричные ключи, открытые в текущем сеансе. Открытые симметричные ключи привязаны к сеансу, а не к контексту безопасности. Открытый ключ останется доступным, пока не будет явно закрыт или сеанс не будет прерван. Инструкция

CLOSE ALL SYMMETRIC KEYS закрывает любой главный ключ базы данных, который был открыт в текущем сеансе при помощи инструкции OPEN MASTER KEY.

6. ENCRYPTBYKEY - производит шифрование данных при использовании симметричного ключа. Функция EncryptByKey использует симметричный ключ. Этот ключ должен быть открыт. Если симметричный ключ уже открыт в текущем сеансе, не нужно открывать его снова в контексте запроса.

Структура проверки подлинности позволяет исключить прямую подмену зашифрованных полей.

Если при шифровании данных указана структура проверки подлинности, то для дешифровки данных функцией DecryptByKey потребуются эти же данная структура проверки подлинности. При шифровании хэш данных проверки подлинности шифруется вместе с данными. При дешифровке эта же структура проверки подлинности должны быть переданы функции DecryptByKey. Если эти данные не совпадают, дешифровка завершится ошибкой. Это будет означать, что значение перемещено с другого места со времени шифрования. В качестве структуры проверки подлинности рекомендуется использовать столбец, содержащий уникальное и неизменное значение. Если значение структуры проверки подлинности изменится, можно утратить доступ к данным.

Симметричное шифрование и расшифровка осуществляются относительно быстро и подходит для работы с большими объемами данных.

7.DECRYPTBYKEY - расшифровывает данные с помощью симметричного ключа. В функции DecryptByKey используется симметричный ключ. Этот симметричный ключ должен быть открыт уже в базе данных. Одновременно могут быть открыты несколько ключей. Открывать ключ непосредственно перед раскодированием необязательно.

Симметричное кодирование и декодирование осуществляется относительно быстро и подходит для работы с большими объемами данных.

Рассмотрим пример шифрования с помощью симметричного ключа. Будем создавать два ключа первый будет шифровать данные, а второй будет шифровать первый ключ.

1. Сначала создаём второй ключ, а потом создаём ключ, который шифрует данные (Рис. 10):

```
USE Библиотека
go
--Создаём симметричный ключ для шифрования другого симметричного ключа
CREATE SYMMETRIC KEY SymKey
WITH ALGORITHM =Triple_DES
ENCRYPTION BY PASSWORD='password'
```

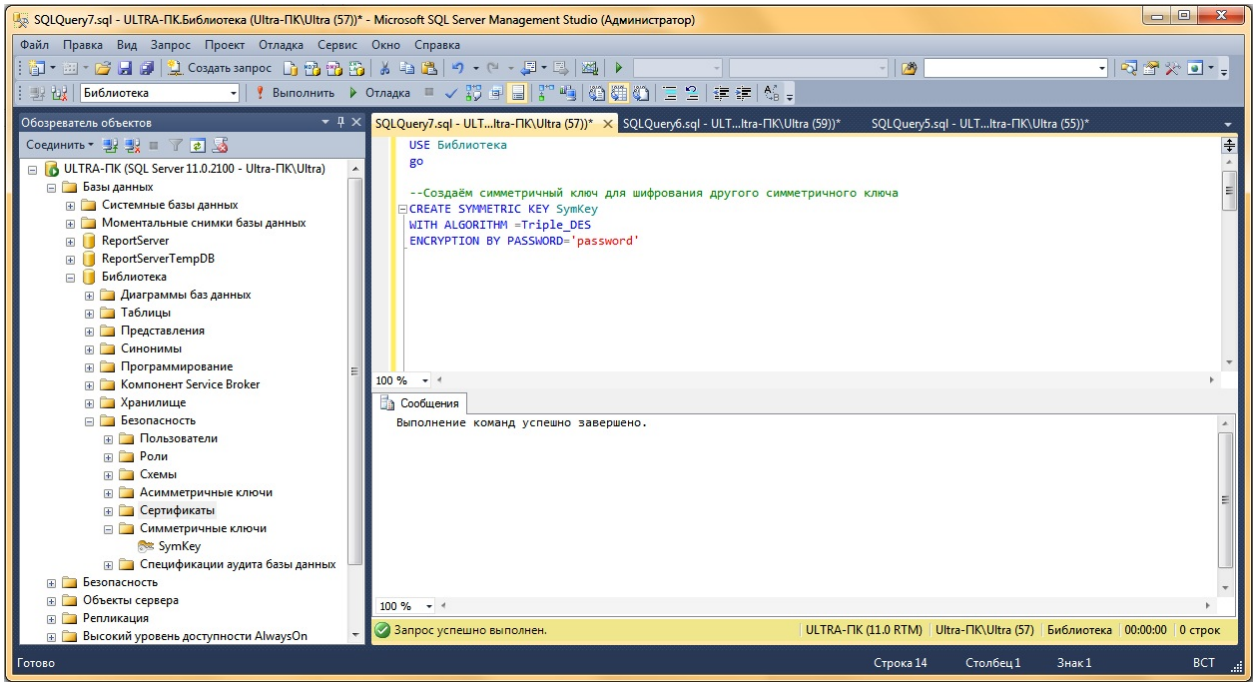

Рис. 10 – Создание ключа для шифрования другого ключа

2. Открываем этот ключ:

```
OPEN SYMMETRIC KEY SymKey
DECRYPTION BY PASSWORD='password'
```

3. Создаём и открываем ключ, который шифрует данные (Рис.11):

```
--Создаём симметричный ключ для шифрования данных
CREATE SYMMETRIC KEY SymData
```

```
WITH ALGORITHM=Triple_DES
ENCRYPTION BY SYMMETRIC KEY SymKey
--Открываем ключ для шифрования данных
OPEN SYMMETRIC KEY SymData
DECRYPTION BY SYMMETRIC KEY SymKey
```

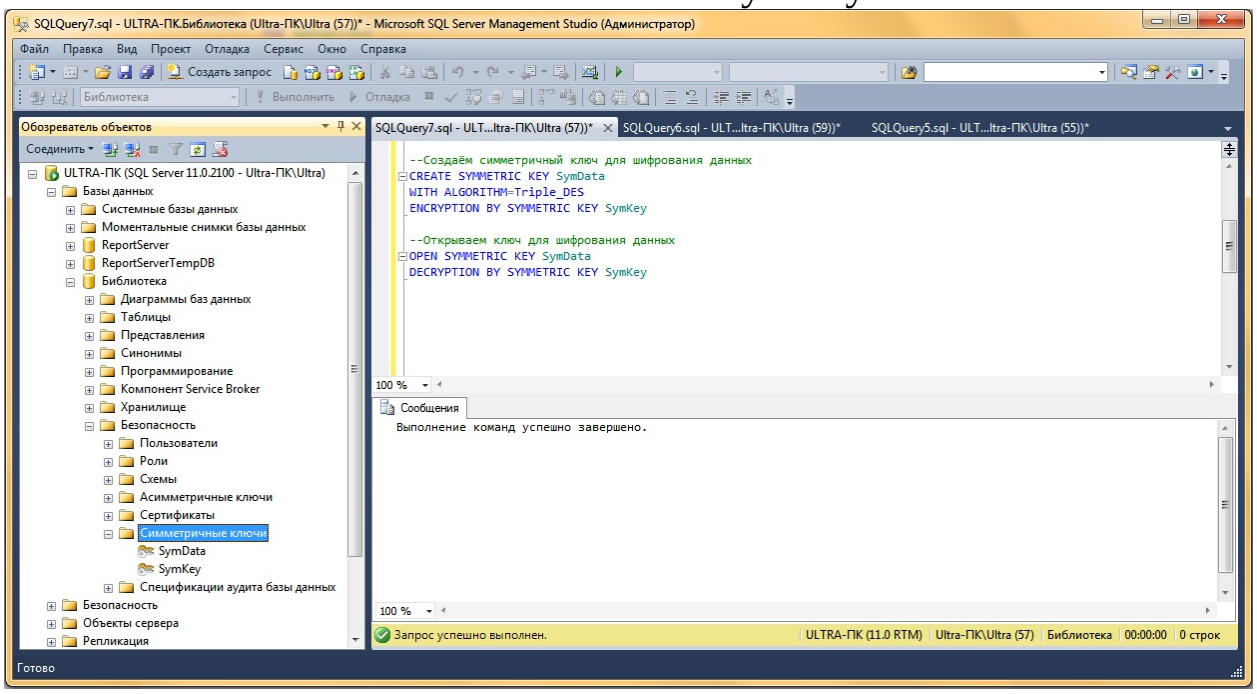

Рис. 11 – Создание ключ для шифрования данных

4. Инициализируем текст, для шифрования, шифруем его и сразу же расшифровываем (Рис. 12):

```
--Инициализируем открытый текст
DECLARE @plaintext nvarchar(512)
SET @plaintext='Добрый день'
print @plaintext

--Шифруем данные
DECLARE @ciphertext varbinary(1024)
SET @ciphertext=ENCRYPTBYKEY
(KEY_GUID('SymData'),@plaintext)
print @ciphertext

--Расшифровываем данные
SET @plaintext=CAST
```

(DECRYPTBYKEY(@ciphertext) as nvarchar(512))
print @plaintext

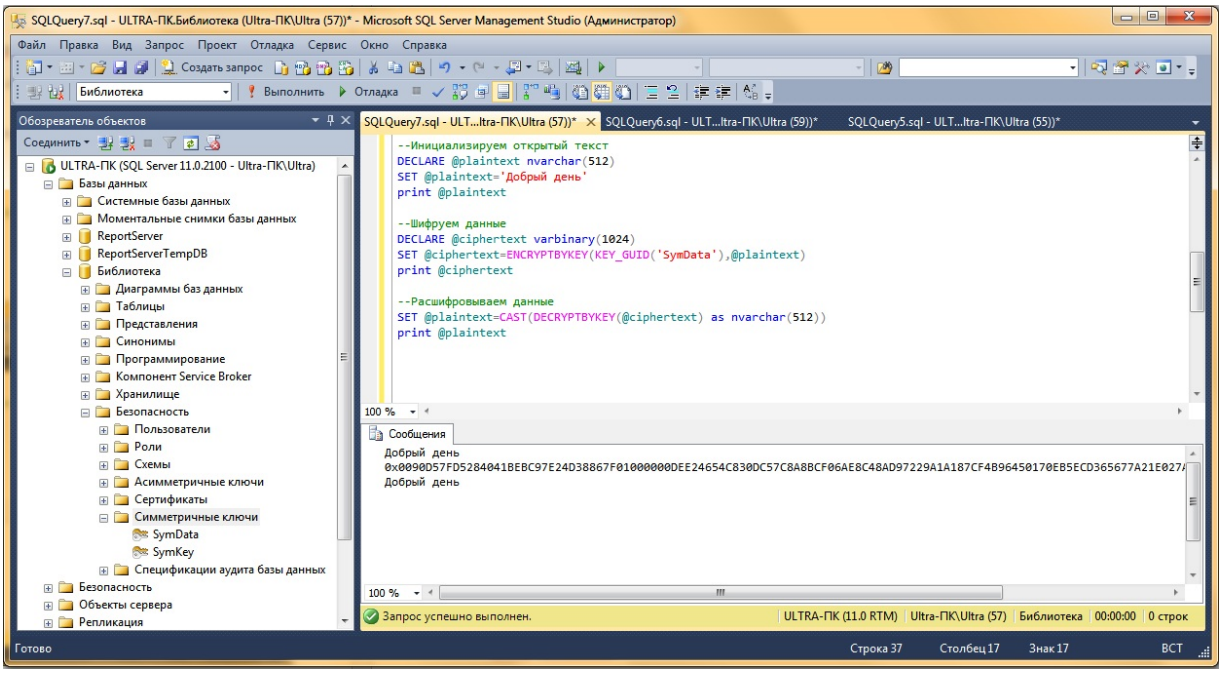

Рис. 12 – Шифрование и дешифрование текста

5. Закрываем ключ шифрования данных и ключ шифрования ключа (Рис. 13):

--Закрываем ключ шифрования данных
CLOSE SYMMETRIC KEY SymData
--Закрываем ключ шифрования ключа
CLOSE SYMMETRIC KEY SymKey

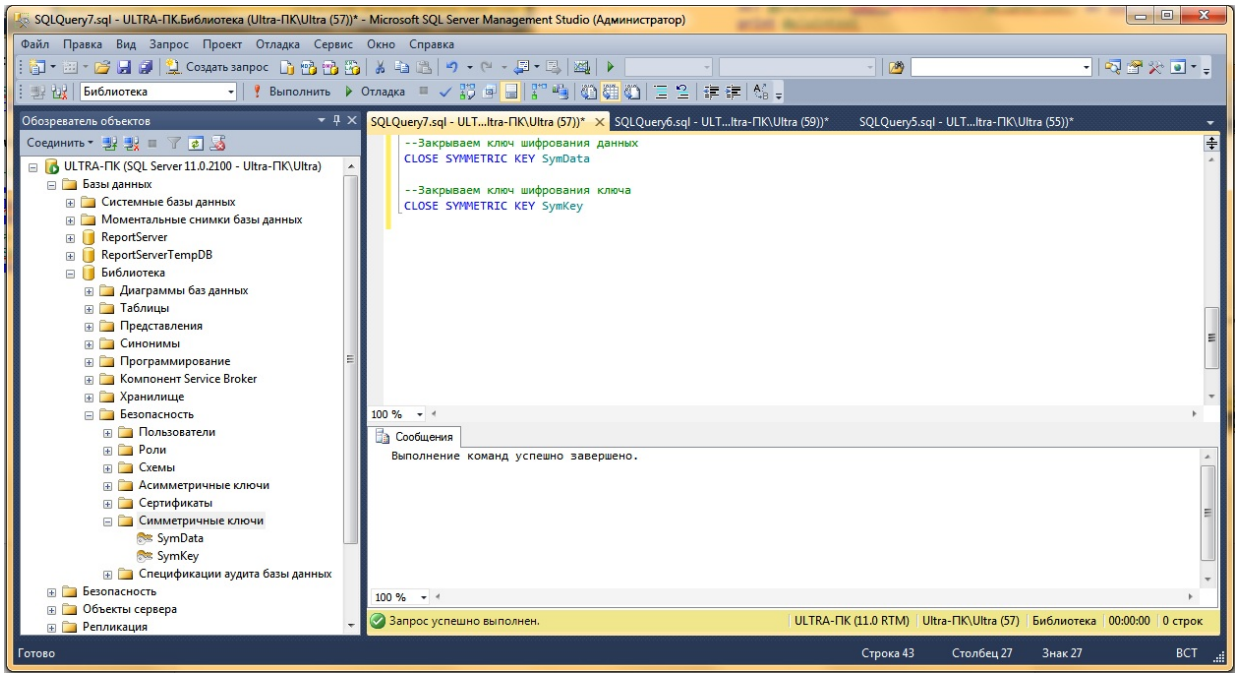

Рис. 13 – Закрываем симметричные ключи

SQL SERVER 2008 позволяет также шифровать данные с использование ключевых фраз (PassPhase). Ключевая фраза - это строка или двоичное значение от которого sql server может наследовать симметричный ключ для шифрования данных.

Функция ENCRYPTBYPASSPHRASE - шифрование данных с помощью парольной фразы с использованием алгоритма TRIPLE DES и 128-битного ключа.

DECRYPTBYPASSPHRASE - расшифровывает данные, зашифрованные с помощью парольной фразы.

Пример (Рис. 14):

```
Declare @plaintext nvarchar(1000), @enc_text varbinary(2000)
SET @plaintext='Я помню чудное мгновенье'
SET @enc_text=ENCRYPTBYPASSPHRASE('LOM',@plaintext)
SELECT 'Оригинальный текст: ',@plaintext
SELECT 'Зашифрованный текст: ',@enc_text
SELECT 'Расшифровка:', CAST (DECRYPTBYPASSPHRASE
('LOM',@enc_text) as nvarchar(1000))
```

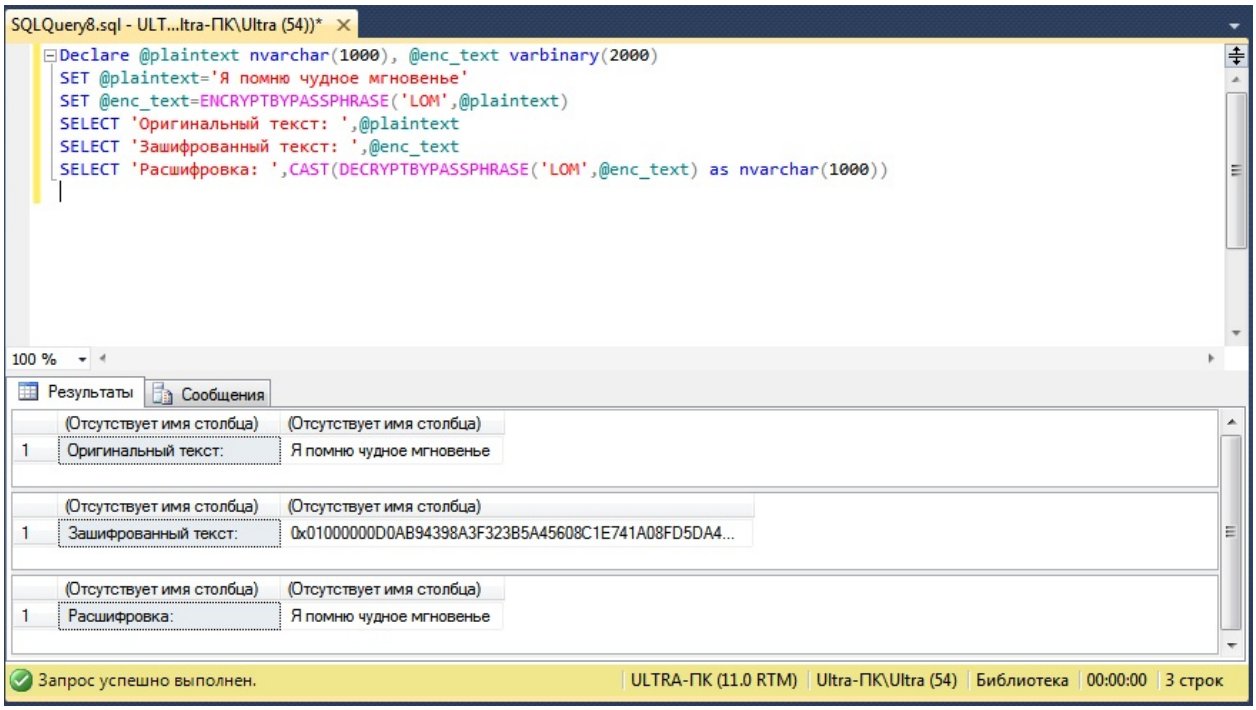

Рис. 14 – Шифрование с помощью ключевой фразы

Создадим базу данных, данные в которой шифруются:

1. с использованием симметричного ключа, который будет зашифрован с помощью асимметричного ключа.
2. с использованием симметричного ключа, но на этот раз симметричный ключ шифруется сертификатом.

Создание базы данных:

```
USE [master]
GO
CREATE DATABASE [DB]
GO
```

Запустите код T-SQL для создания таблицы с именем TelephonNumber в базе данных DB:

```
USE [DB]
GO
CREATE TABLE [dbo].[ TelephonNumber]
([PersonID] [int] PRIMARY KEY,
[TelephonNumber] [varbinary](max))
```

```
GO
```

Эта таблица будет содержать ложную информацию о телефонных номерах. Номера телефонов будут сохранены в столбце двоичных переменных, потому что они будут шифроваться.

Используйте следующий программный код для создания главного ключа DMK базы данных DB, шифруемого с помощью парольной фразы $tr0nGPa$$w0rd:

```
USE [DB]
GO
CREATE MASTER KEY ENCRYPTION BY PASSWORD = '$tr0nGPa$$w0rd'
GO
```

Необходимо создать асимметричный ключ, зашифровать его парольной фразой $tr0nGPa$$w0rd, создать симметричный ключ и зашифровать симметричный ключ с помощью только что созданного асимметричного ключа. Выполнить эти задачи можно, запустив программный код:

```
USE [DB]
GO
--Создание асимметричного ключа, зашифрованного парольной фразой StrongPa$$w0rd!
CREATE ASYMMETRIC KEY MyAsymmetricKey
WITH ALGORITHM = RSA_2048
ENCRYPTION BY PASSWORD = 'StrongPa$$w0rd!'
GO
--Создание симметричного ключа, зашифрованного асимметричным ключом
CREATE SYMMETRIC KEY MySymmetricKey
WITH ALGORITHM = AES_256
ENCRYPTION BY ASYMMETRIC KEY MyAsymmetricKey
GO
```

1. Теперь мы можем приступить к шифрованию данных. Для этого необходимо сначала открыть симметричный ключ, только

что созданный с помощью команды OPEN SYMMETRIC KEY, за которой следует имя симметричного ключа. Затем указать, что нужно расшифровать его с использованием заданного асимметричного ключа. Программный код выглядит следующим образом:

```
USE [DB]
GO
OPEN SYMMETRIC KEY MySymmetricKey
DECRYPTION BY ASYMMETRIC KEY MyAsymmetricKey
WITH PASSWORD = 'StrongPa$$w0rd!'
GO
```

После выполнения этого кода направьте запрос в представление sys.openkeys, чтобы убедиться, что ключ открыт (Рис.15):

```
USE [DB]
GO
SELECT * FROM [sys].[openkeys]
```

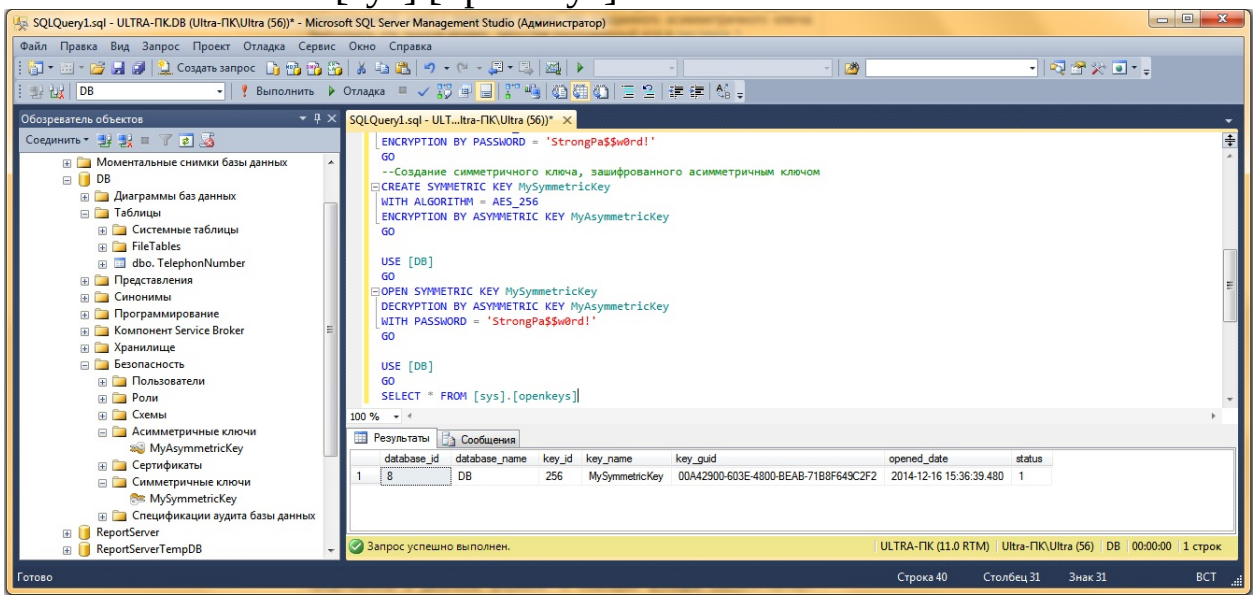

Рис. 15 – Открытие симметричного ключа

Введем несколько телефонных номеров в таблицу TelephoneNumber, запустив код:

```
USE [DB]
GO
```

```
DECLARE @SymmetricKeyGUID AS [uniqueidentifier]
SET @SymmetricKeyGUID = KEY_GUID('MySymmetricKey')
IF (@SymmetricKeyGUID IS NOT NULL)
BEGIN
INSERT INTO [dbo].[ TelephonNumber]
VALUES (01, ENCRYPTBYKEY(@SymmetricKeyGUID,
N'8-761-123-87-63'))
INSERT INTO [dbo].[ TelephonNumber]
VALUES (02, ENCRYPTBYKEY(@SymmetricKeyGUID,
N'8-768-765-87-65'))
INSERT INTO [dbo].[ TelephonNumber]
VALUES (03, ENCRYPTBYKEY(@SymmetricKeyGUID,
N'8-761-234-11-11'))
END
TRUNCATE TABLE [dbo].[TelephonNumber]
```

Выведем получившуюся таблицу на экран (Рис. 16):

```
USE [DB]
GO
SELECT * FROM [dbo].[ TelephonNumber]
```

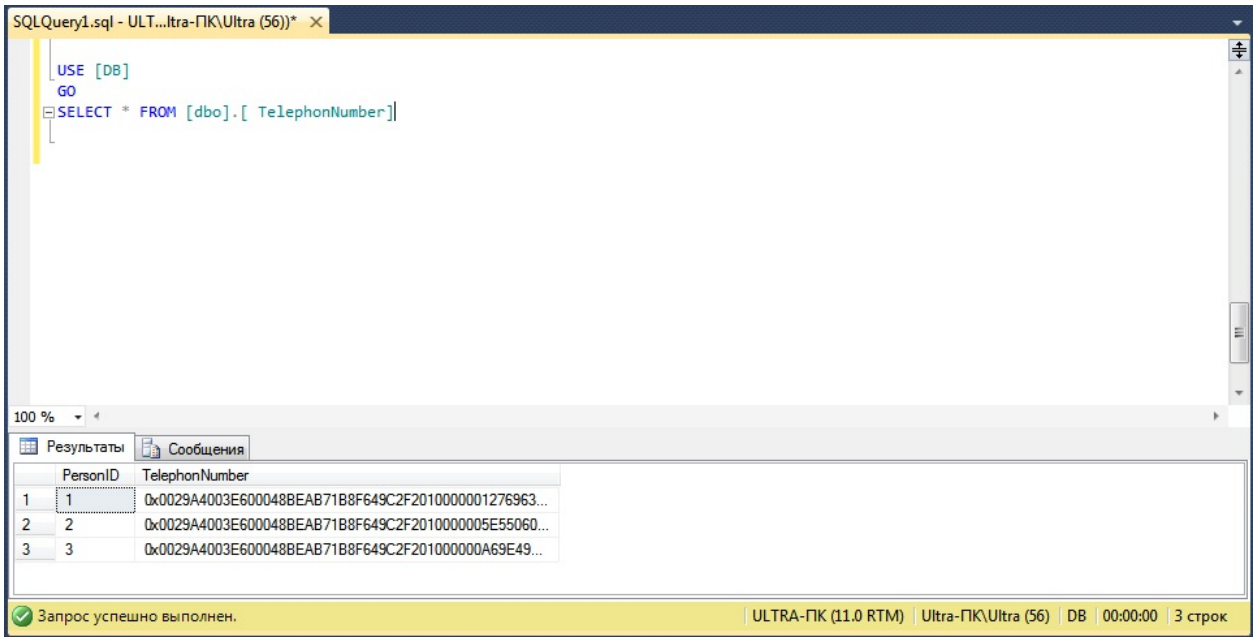

Рис. 16 – Зашифрованные данные из таблицы TelephoneNumber

Все данные в столбце TelephoneNumber представлены в двоичном формате. С помощью функции DECRYPTBYKEY можно просмотреть зашифрованные данные (Рис. 17):

```
USE [DB]
GO
SELECT [PersonID],
CONVERT([nvarchar](32),
DECRYPTBYKEY(TelephonNumber))
AS [TelephonNumber]
FROM [dbo].[ TelephonNumber]
GO
```

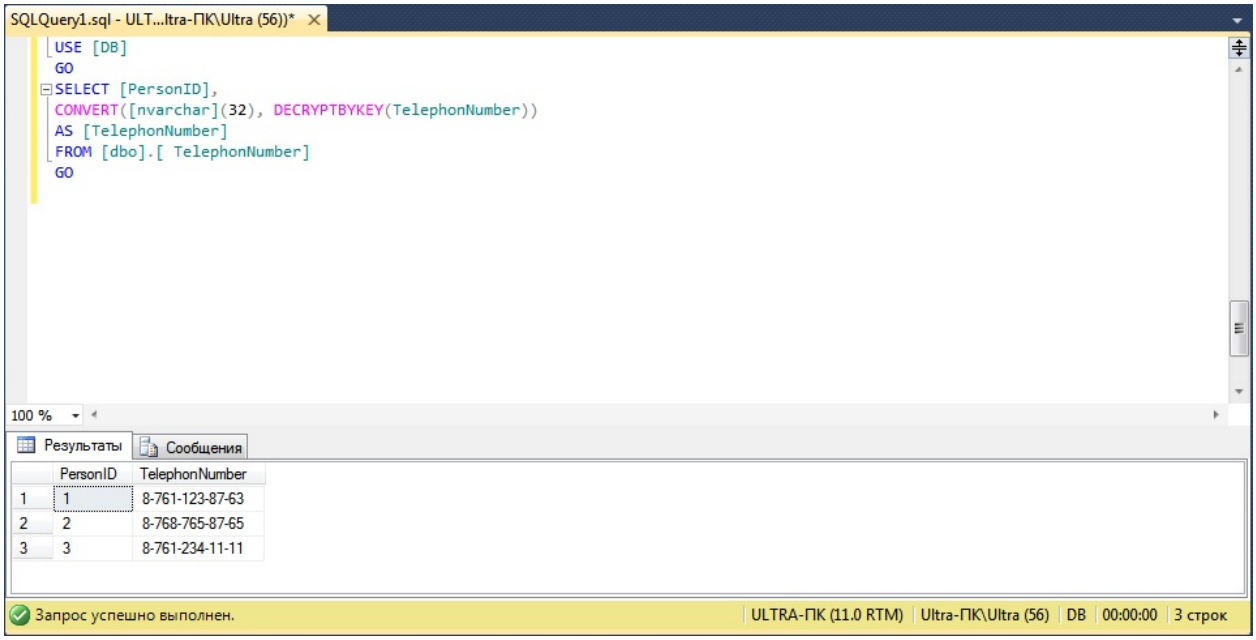

Рис. 17– Просмотр зашифрованных данных

2. Создадим сертификат с помощью инструкции CREATE CERTIFICATE. Затем создается симметричный ключ, шифруемый сертификатом. Наконец, открыв симметричный ключ, код вставляет три строки в таблицу TelephonNumber:

```
USE [DB]
```

```
GO
--Создание сертификата
CREATE CERTIFICATE
[CertToEncryptSymmetricKey]
WITH SUBJECT ='Самозаверяющий сертификат
для шифрования симметричного ключа.'
--Создание симметричного ключа, зашифрованного сертификатом
CREATE SYMMETRIC KEY
[SymmetricKeyEncryptedWithCert]
WITH ALGORITHM = AES_256
ENCRYPTION BY CERTIFICATE
[CertToEncryptSymmetricKey]
--Открытие симметричного ключа
OPEN SYMMETRIC KEY
[SymmetricKeyEncryptedWithCert]
DECRYPTION BY CERTIFICATE
[CertToEncryptSymmetricKey]
--Усечение таблицы TelephonNumber
TRUNCATE TABLE
[dbo].[ TelephonNumber]
--Вставка данных в таблицу
DECLARE @SymmetricKeyGUID
AS [uniqueidentifier]
SET @SymmetricKeyGUID =
KEY_GUID
('SymmetricKeyEncryptedWithCert')
IF (@SymmetricKeyGUID IS NOT NULL)
BEGIN
INSERT INTO [dbo].[ TelephonNumber]
VALUES (01, ENCRYPTBYKEY
(@SymmetricKeyGUID,
N'8-861-123-87-63'))
INSERT INTO [dbo].[ TelephonNumber]
VALUES (02, ENCRYPTBYKEY
(@SymmetricKeyGUID,
N'8-868-765-87-65'))
INSERT INTO [dbo].[ TelephonNumber]
```

```
VALUES (03, ENCRYPTBYKEY
(@SymmetricKeyGUID,
N'8-861-234-11-11'))
END
```

Просмотр зашифрованных данных (Рис. 18):

```
USE [DB]
GO
SELECT * FROM
[dbo].[ TelephonNumber]
```

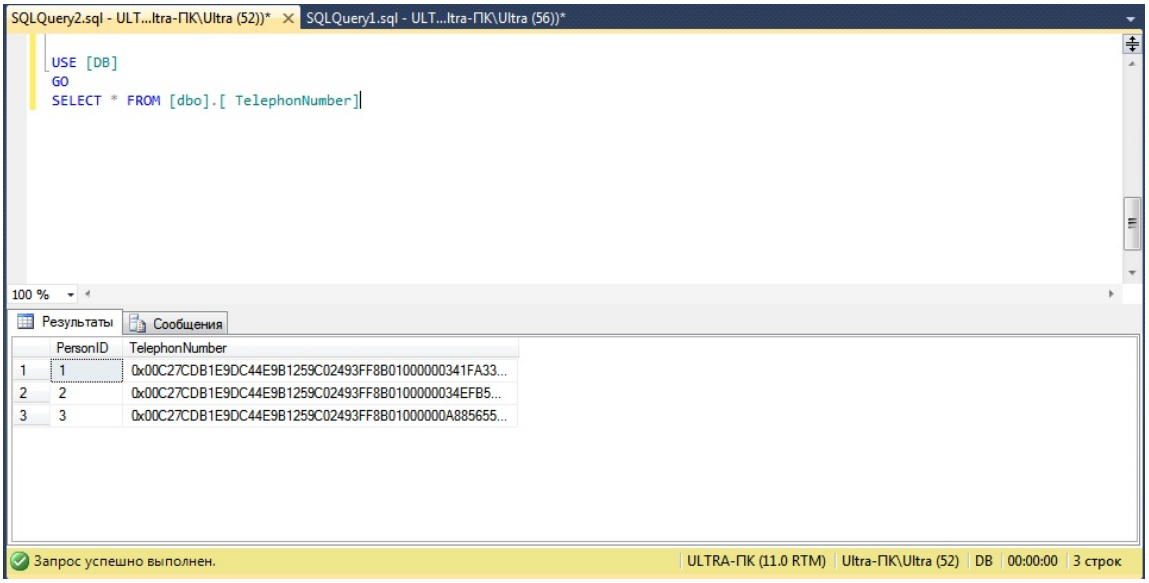

Рис. 18 – Вывод данных таблицы “TelephonNumber”

Расшифровка (Рис. 19):

```
USE [DB]
GO
SELECT [PersonID],
CONVERT([nvarchar](32), DECRYPTBYKEY(TelephonNumber))
AS [TelephonNumber]
FROM [dbo].[ TelephonNumber]
GO
```

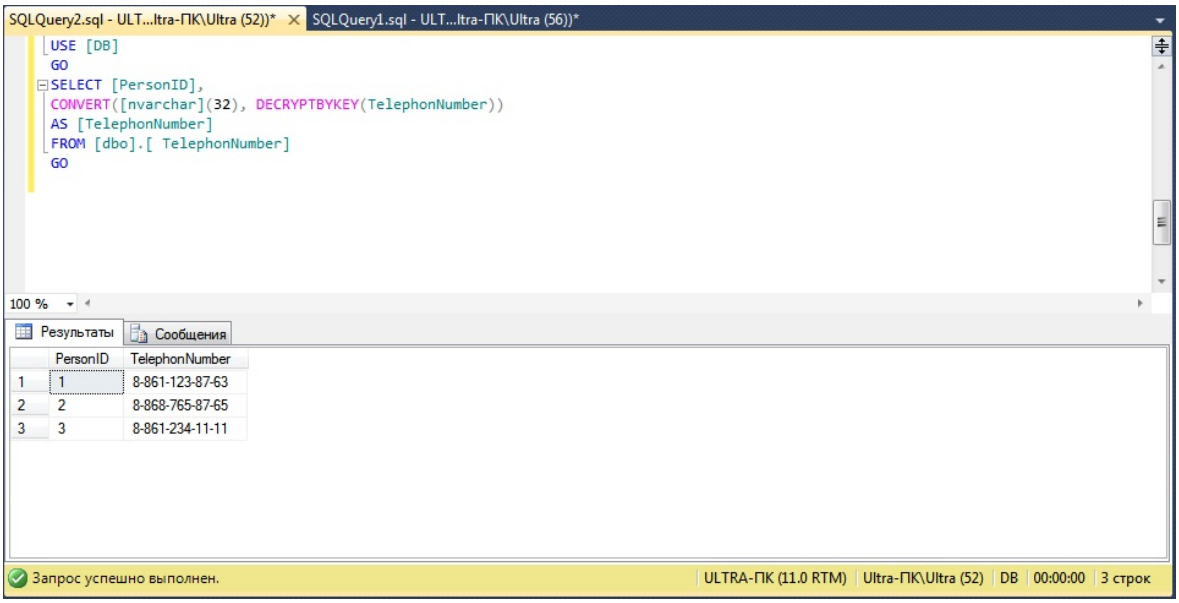

Рис. 19 – Дешифровка данных таблицы “TelephonNumber”

Рассмотрим пример прозрачного шифрования данных (Рис. 20)

1. Создаем главный ключ шифрования

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

Созданный наш ключ можно увидеть в view:

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

2. Чтобы удалить ключ введите следующий запрос:

```
drop master key
```

3. После того как создали главный ключ, необходимо сделать его резервную копию и поместить резервную копию в надежное место:

```
BACKUP MASTER KEY TO FILE = 'C:\Cert\MasterBackup.bak'
ENCRYPTION BY PASSWORD = 'Password1'
```

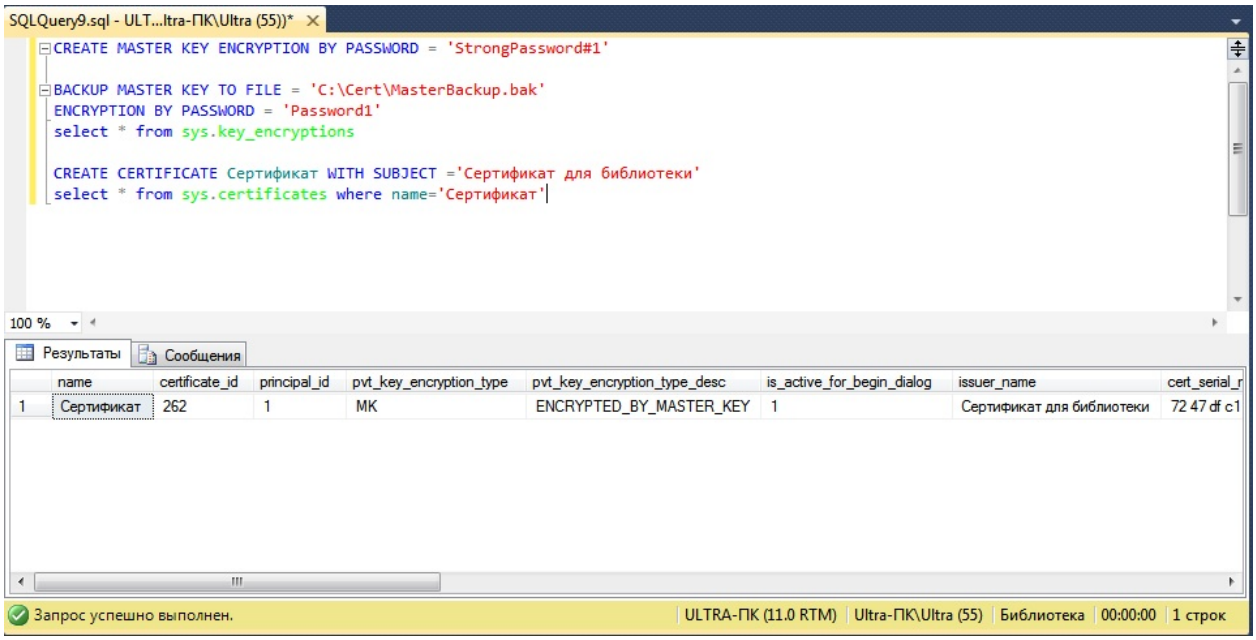

Рис. 20 – Создание главного ключа и сертификата

4. Создание сертификата осуществляется следующим запросом:

```
CREATE CERTIFICATE Сертификат
WITH SUBJECT ='Сертификат для библиотеки'
```

Проверка наличия созданного сертификата:

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

5. Создание резервной копии сертификата с закрытым ключом (Рис. 21):

```
BACKUP CERTIFICATE Сертификат
TO FILE = 'C:\Cert\Certif'
WITH PRIVATE KEY
(FILE = 'C:\Cert\PrivateCertif',
 ENCRYPTION BY PASSWORD = 'Password#3');
```

Выполнив всё вышеизложенное приступим к созданию ключа шифрования в нашей базе данных с использование нашего сертификата:

```
USE Библиотека
go
CREATE DATABASE ENCRYPTION KEY
WITH ALGORITHM = AES_128
ENCRYPTION BY SERVER CERTIFICATE Сертификат;
```

Включаем шифрование для нашей базы данных:

```
ALTER DATABASE Библиотека
SET ENCRYPTION ON;
```

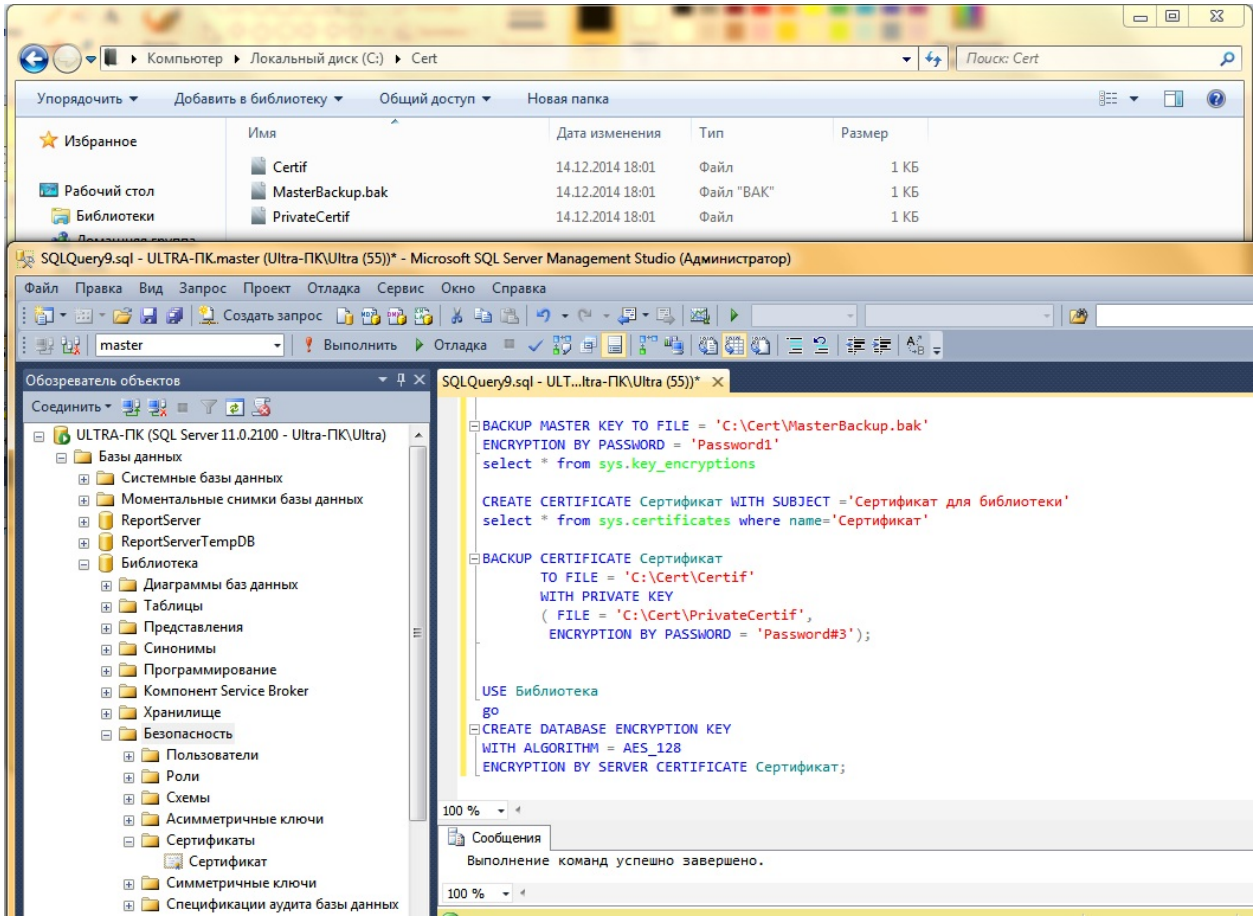

Рис. 21 – Резервные копии сертификата и главного ключа. Создание ключа шифрования в нашей базе данных

Чтобы посмотреть зашифрованные базы данных, нужно ввести следующий запрос (Рис. 22):

```
USE [master]
GO SELECT db.[name]
, db.[is_encrypted]
, dm.[encryption_state]
, dm.[percent_complete]
, dm.[key_algorithm]
, dm.[key_length]
FROM [sys].[databases] db
LEFT OUTER JOIN [sys].[dm_database_encryption_keys] dm
ON db.[database_id] = dm.[database_id];
 GO
```

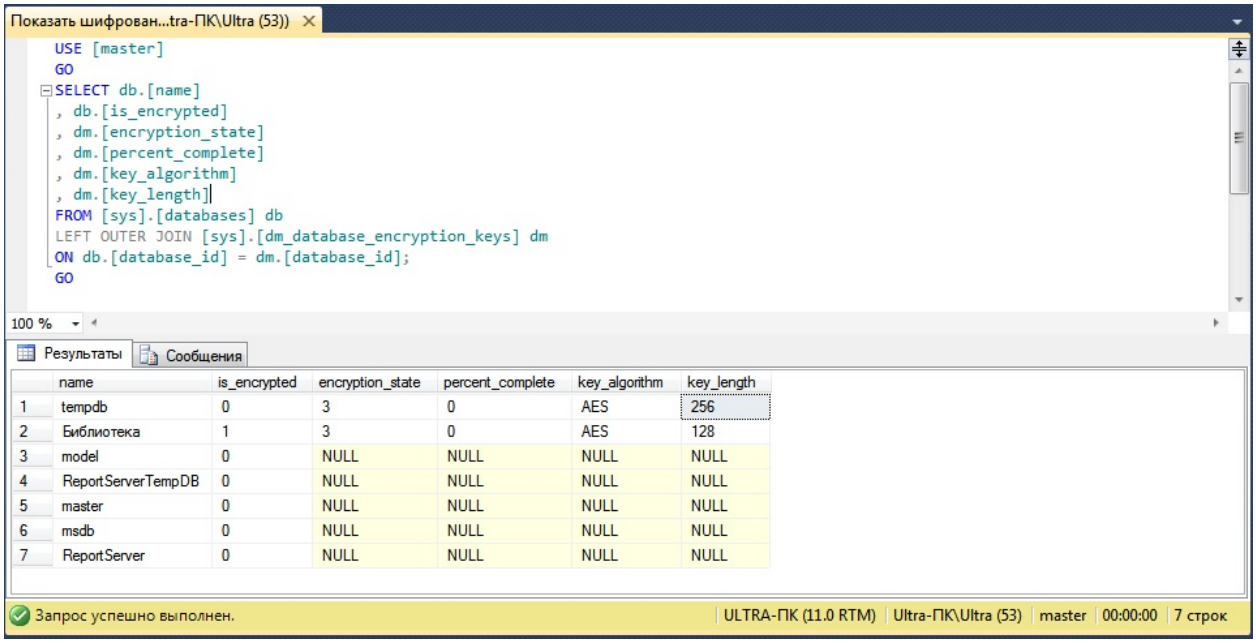

Рис. 22 – Вывод списка баз данных

Несколько моментов в работе с базой данных с включенным шифрованием:

Если создать резервную копию и попытаться восстановить на другом сервере, то получим ошибку:

```
Msg 33111, Level 16, State 3, Line 2
Cannot find server certificate with thumbprint
'0x5B139FF1F2C5ED9EB3D503E78A63DEF3DD1FD96F'.
Msg 3013, Level 16, State 1, Line 2
```

RESTORE DATABASE is terminating abnormally.

Такую же ошибку получим и при попытки присоединения файлов базы данных. Порядок восстановления базы данных с прозрачным шифрованием на другом экземпляре MS SQL Server:
1. Создать мастер главный ключ шифрования на сервере MS SQL Server.

2. Восстановить из резервной копии сертификат с закрытым ключом:

```
CREATE CERTIFICATE Сертификат
FROM FILE = 'c:\Program Files\Microsoft SQL
Server\MSSQL11.MSSQLSERVER\MSSQL\DATA\Certif'
WITH PRIVATE KEY (FILE = 'c:\Program Files\Microsoft SQL
Server\MSSQL11.MSSQLSERVER\MSSQL\DATA\PrivateCertif',
DECRYPTION BY PASSWORD = 'Password#3');
```

3. Восстановить шифрованную базу данных или присоединить файлы базы данных с включенным шифрованием. БД готова для работы.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. http://www.osp.ru/win2000/2013/05/13035359/#list2
2. http://dbasimple.blogspot.ru/2013/08/ms-sql-server.html
3. Microsoft SQL Server 2008. Руководство для начинающих
4. http://rutube.ru/video/fa041b99e956c1534a6a424e2a9aac57/
5. http://msdn.microsoft.com/ru-ru/library/bb510663.aspx