Аутентификация и шифрование данных БД

Основные понятия

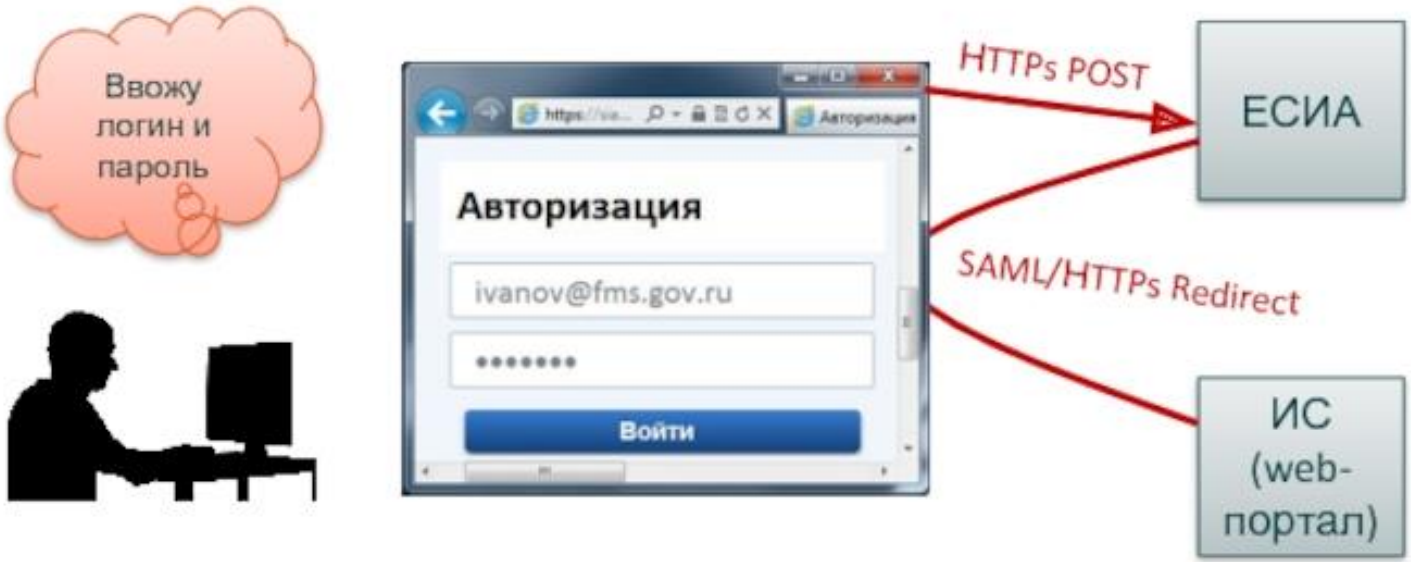
Идентификация – процедура распознавания субъекта (пользователя, процесса, действующего от имени пользователя, аппаратно-программного компонента) по его уникальному идентификатору, присвоенному субъекту ранее и занесенному в базу данных в момент его регистрации как легального пользователя системы.

Аутентификация – процедура проверки подлинности входящего в систему субъекта, предъявившего свой идентификатор. Аутентификацию не следует путать с авторизацией (процедурой предоставления субъекту определённых прав).

Аутентификация заключается в предоставлении ответа на следующий вопрос: "Имеет ли данный пользователь законное право на доступ к системе?" Таким образом, данная концепция безопасности определяет процесс проверки подлинности учетных данных пользователя, чтобы не допустить использование системы несанкционированными пользователями.

Аутентификацию можно реализовать путем запроса, требуя, чтобы пользователь предоставил, например, следующее:

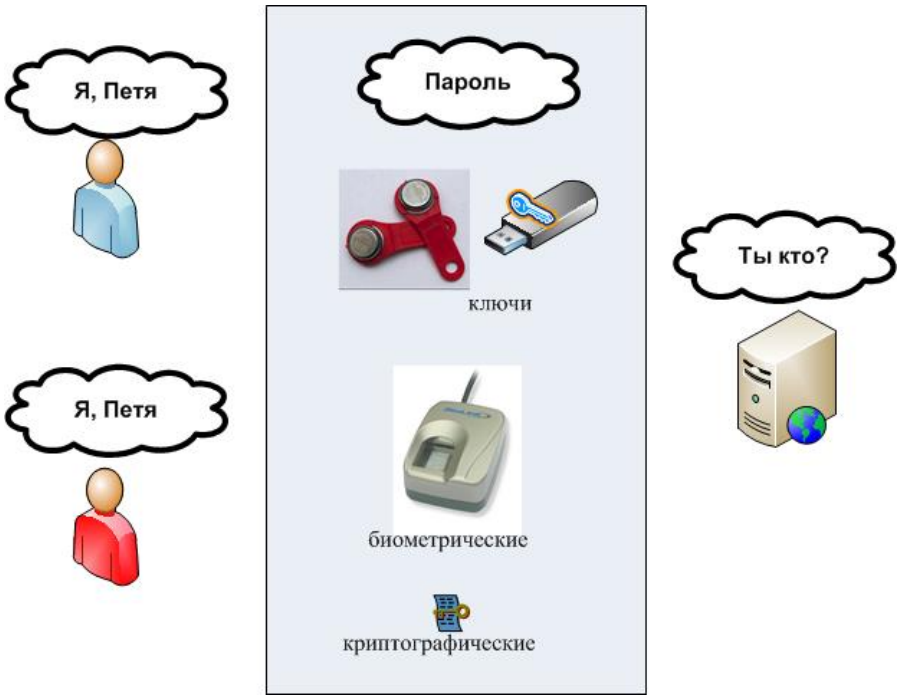
- нечто, что известно пользователю (обычно пароль);
- нечто, что принадлежит пользователю, например, идентификационную карту;
- физические характеристики пользователя, например, подпись или отпечатки пальцев



1. Пользователь обращается к защищённому ресурсу ИС
2. ИС перенаправляет пользователя в ЕСИА
3. Пользователь проходит аутентификацию в ЕСИА
4. ЕСИА передаёт в ИС утверждения о пользователе и перенаправляет пользователя в ИС

Методы аутентификации:

- **парольные** (PIN коде и т.д.) - уникальная последовательность символов, которую пользователь должен знать.
- **"ключе"** - в случае электронных систем это электронный ключ, который хранится на носителе (смарт-карты, электронные таблетки iButton, USB-токены и т. д.)
- **биометрические** (отпечаток пальца, рисунок радужной оболочки глаза, форма лица, параметры голоса и т. д.)
- **криптографические**



- Наиболее применяемый способ подтверждения аутентификации реализуется посредством использования имени пользователя и пароля. Система проверяет достоверность этой информации, чтобы решить, имеет ли данный пользователь законное право на доступ к системе или нет. Этот процесс может быть усилен применением шифрования.
- ***Шифрование данных*** представляет собой процесс кодирования информации таким образом, что она становится непонятной, пока не будет расшифрована пользователем.

Аутентификация

Система безопасности компонента Database Engine состоит из двух разных подсистем безопасности:

- системы безопасности Windows;
- системы безопасности SQL Server.

Система безопасности Windows

Система безопасности Windows определяет безопасность на уровне операционной системы, т.е. метод, посредством которого пользователи входят в систему Windows, используя свои учетные записи Windows. (Аутентификация посредством этой подсистемы также называется аутентификацией Windows.)

Система безопасности SQL Server

Система безопасности SQL Server определяет дополнительную безопасность, требуемую на уровне системы баз данных, т.е. способ, посредством которого пользователи, уже вошедшие в операционную систему, могут подключаться к серверу базы данных.

Система безопасности SQL Server определяет регистрационное имя входа (или просто называемое логином) в SQL Server, которое создается в системе и ассоциируется с определенным паролем. Некоторые регистрационные имена входа в SQL Server идентичны существующим учетным записям Windows. (Аутентификация посредством этой подсистемы также называется аутентификацией SQL Server.)

На основе этих двух подсистем безопасности компонент Database Engine может работать в одном из следующих режимов аутентификации:

- в режиме Windows и
- в смешанном режиме.

Режим Windows требует, чтобы пользователи входили в систему баз данных исключительно посредством своих учетных записей Windows. Система принимает данные учетной записи пользователя, полагая, что они уже были проверены и одобрены на уровне операционной системы. Такой способ подключения к системе баз данных называется доверительным соединением (trusted connection), т.к. система баз данных доверяет, что операционная система уже проверила подлинность учетной записи и соответствующего пароля.

- **Смешанный режим позволяет** пользователям подключаться к компоненту Database Engine посредством аутентификации Windows или аутентификации SQL Server. Это означает, что некоторые учетные записи пользователей можно настроить для использования подсистемы безопасности Windows, а другие, вдобавок к этому, могут использовать также и подсистему безопасности SQL Server.
- Аутентификация SQL Server предоставляется исключительно в целях обратной совместимости. Поэтому зачастую следует использовать аутентификацию Windows.

Реализация режима аутентификации

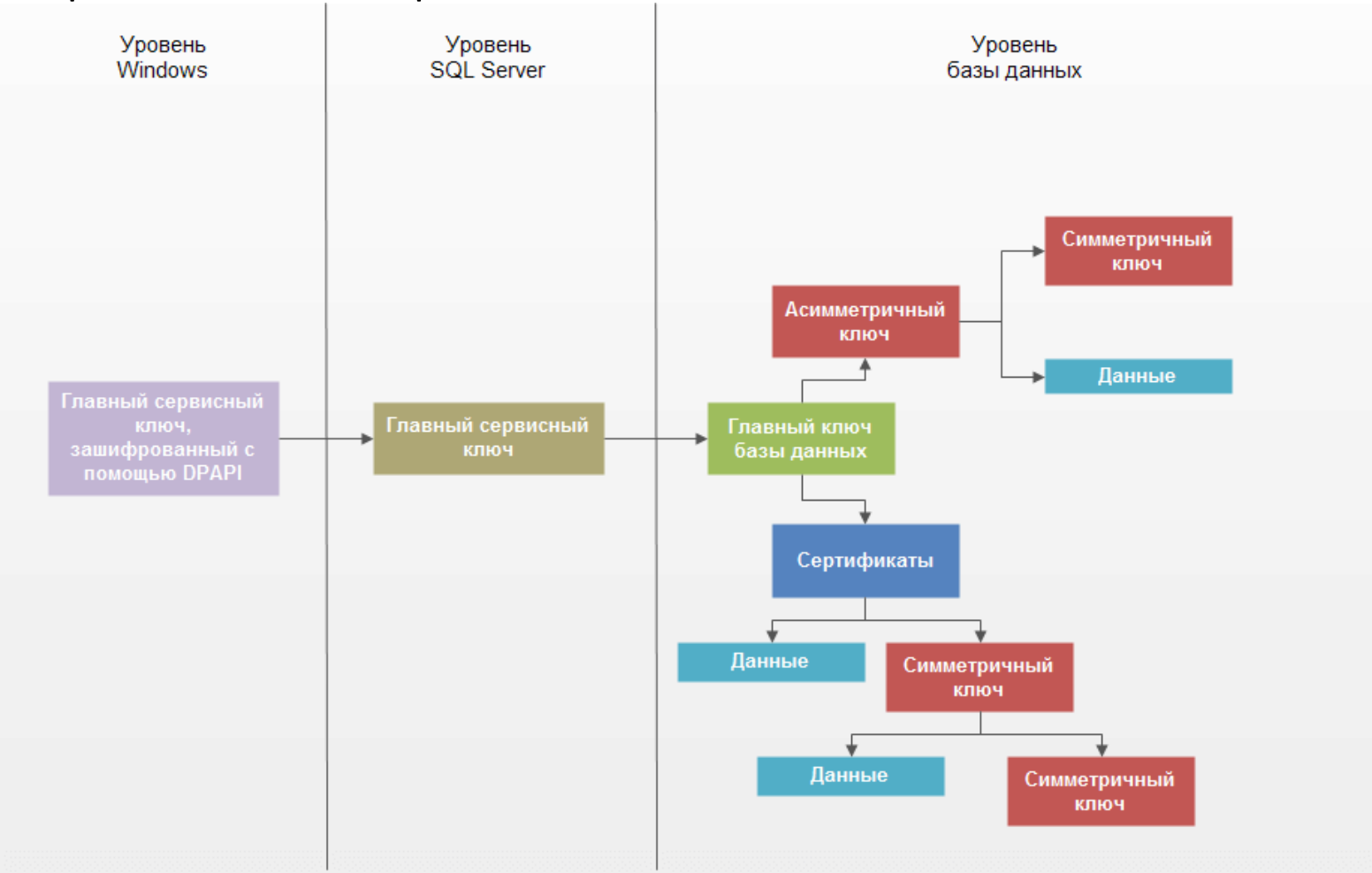
- Выбор одного из доступных режимов аутентификации осуществляется посредством среды SQL Server Management Studio. Чтобы установить режим аутентификации Windows, щелкните правой кнопкой сервер баз данных в окне Object Explorer и в контекстном меню выберите пункт Properties. Откроется диалоговое окно Server Properties, в котором нужно выбрать страницу Security, а на ней Windows Authentication Mode. Для выбора смешанного режима в этом же диалоговом окне Server Properties вам нужно выбрать Server and Windows Authentication Mode.
- После успешного подключения пользователя к компоненту Database Engine его доступ к объектам базы данных становится независимым от использованного способа аутентификации для входа в базу данных - аутентификации Windows или SQL Server аутентификации.

Шифрование данных

- Шифрование - это процесс приведения данных в запутанное непонятное состояние, вследствие чего повышается уровень их безопасности.

Обычно конкретная процедура шифрования осуществляется с использованием определенного алгоритма. Наиболее важный алгоритм шифрования называется **RSA**, по первым буквам фамилий его создателей - Rivers, Shamir и Adelman.

Компонент Database Engine обеспечивает безопасность данных посредством иерархии уровней шифрования и инфраструктуры управления ключами. Каждый уровень защищает следующий за ним уровень шифрования, используя комбинацию сертификатов, асимметричных и симметричных ключей:



- На рисунке выше главный сервисный ключ задает ключ, который управляет всеми другими ключами и сертификатами. Главный сервисный ключ создается автоматически при установке компонента Database Engine. Этот ключ зашифрован с помощью *API-интерфейса защиты данных Windows (DPAPI - Data Protection API)*.
- Важным свойством главного сервисного ключа является то, что он управляется системой. Хотя системный администратор может выполнять разные задачи по обслуживанию ключа, ему следует выполнять лишь одну из них - резервное копирование главного сервисного ключа, чтобы его можно было восстановить в случае повреждения.
- Как можно видеть на рисунке, главный сервисный ключ базы данных является корневым объектом шифрования на уровне базы данных для всех ключей, сертификатов и данных. Каждая база данных имеет один главный ключ базы данных, который создается посредством инструкции **CREATE MASTER KEY**. Поскольку главный ключ базы данных защищен главным сервисным ключом системы, то система может автоматически расшифровывать главный ключ базы данных.

Существующий главный ключ базы данных можно использовать для создания пользовательских ключей.

Существует три формы пользовательских ключей:

- симметричные ключи;
- асимметричные ключи;
- сертификаты

Симметричные ключи

- В системе шифрования с использованием симметричного ключа оба участника обмена - отправитель и получатель сообщения - применяют один и тот же ключ. Иными словами, для шифрования информации и ее обратного расшифровывания используется этот единственный ключ.
- Использование симметричных ключей имеет много преимуществ и один недостаток. Одним из преимуществ использования симметричных ключей является то обстоятельство, что с их помощью можно защитить значительно больший объем данных, чем с помощью двух других типов ключей. Кроме этого, использование ключей этого типа намного быстрее, чем использование несимметричных ключей.
- Но с другой стороны, использование симметричного ключа в среде распределенной системы может сделать задачу обеспечения целостности шифрования почти невыполнимой, поскольку один и тот же ключ применяется на обоих концах обмена данными. Таким образом, основной рекомендацией является то, что симметричные ключи должны использоваться только с теми приложениями, в которых зашифрованные данные сохраняются в одном месте.
- Язык Transact-SQL поддерживает несколько инструкций и системных функций применительно к симметричным ключам. В частности, для создания симметричного ключа применяется инструкция **CREATE SYMMETRIC KEY**, а для удаления существующего симметричного ключа - инструкция **DROP SYMMETRIC KEY**. Прежде чем симметричный ключ можно использовать для шифрования данных или для защиты другого ключа, его нужно открыть. Для этого используется инструкция **OPEN SYMMETRIC KEY**.
- После того как вы откроете симметричный ключ, вам нужно для шифрования использовать системную функцию **EncryptByKey**. Эта функция имеет два входных параметра: идентификатор ключа и текст, который требуется зашифровать. Для расшифровки зашифрованной информации применяется системная функция **DecryptByKey**.

Асимметричные ключи

- В случае если у вас имеется распределенная система или если использование симметричных ключей не обеспечивает достаточного уровня безопасности данных, то следует использовать асимметричные ключи. ***Асимметричный ключ*** состоит из двух частей: личного закрытого ключа (private key) и соответствующего общего открытого ключа (public key). Каждый из этих ключей может расшифровывать данные, зашифрованные другим ключом. Благодаря наличию личного закрытого ключа асимметричное шифрование обеспечивает более высокий уровень безопасности данных, чем симметричное шифрование.
- Язык Transact-SQL поддерживает несколько инструкций и системных функций применительно к асимметричным ключам. В частности, для создания нового асимметричного ключа применяется инструкция **CREATE ASYMMETRIC KEY**, а для изменения свойств асимметричного ключа используется инструкция **ALTER ASYMMETRIC KEY**. Для удаления асимметричного ключа применяется инструкция **DROP ASYMMETRIC KEY**.
- После того как вы создали асимметричный ключ, для шифрования данных используйте системную функцию **EncryptByAsymKey**. Эта функция имеет два входных параметра: идентификатор ключа и текст, который требуется зашифровать. Для расшифровки информации, зашифрованной с использованием асимметричного ключа, применяется системная функция **DecryptByAsymKey**.

Сертификаты

- Сертификат открытого ключа, или просто сертификат, представляет собой предложение с цифровой подписью, которое привязывает значение открытого ключа к определенному лицу, устройству или службе, которая владеет соответствующим открытым ключом. Сертификаты выдаются и подписываются *центром сертификации (Certification Authority - CA)*. Сущность, которая получает сертификат из центра сертификации (CA), является субъектом данного сертификата (certificate subject).
- Между сертификатами и асимметричными ключами нет значительной функциональной разницы. Как первый, так и второй используют алгоритм RSA. Основная разница между ними заключается в том, что асимметричные ключи создаются вне сервера.

Сертификаты содержат следующую информацию:

- значение открытого ключа субъекта;
- информацию, идентифицирующую субъект;
- информацию, идентифицирующую издателя сертификата;
- цифровую подпись издателя сертификата.

- Основным достоинством сертификатов является то, что они освобождают хосты от необходимости содержать набор паролей для отдельных субъектов. Когда хост, например, безопасный веб-сервер, обозначает издателя как надежный центр авторизации, этот хост неявно доверяет, что данный издатель выполнил проверку личности субъекта сертификата.
- Сертификаты предоставляют самый высший уровень шифрования в модели безопасности компонента Database Engine. Алгоритмы шифрования с использованием сертификатов требуют большого объема процессорных ресурсов. По этой причине сертификаты следует использовать при реальной необходимости.

Наиболее важной инструкцией применительно к сертификатам является **инструкция CREATE CERTIFICATE**. Использование этой инструкции показано в примере ниже:

```sql
USE SampleDb;

CREATE MASTER KEY
    ENCRYPTION BY PASSWORD = '12345!'

GO
CREATE CERTIFICATE cert01
    WITH SUBJECT = 'Сертификат для схемы dbo';
```

Чтобы создать сертификат без параметра ENCRYPTION BY, сначала нужно создать главный ключ базы данных. (Все инструкции CREATE CERTIFICATE, которые не содержат этот параметр, защищаются главным ключом базы данных.) По этой причине первой инструкцией в примере выше является инструкция CREATE MASTER KEY. После этого инструкция CREATE CERTIFICATE используется для создания нового сертификата cert01, владельцем которого является объект dbo базы данных SampleDb, если этот объект является текущим пользователем.

Редактирование пользовательских ключей

Наиболее важными представлениями каталога применительно к шифрованию являются следующие:

- **sys.symmetric_keys**
- **sys.asymmetric_keys**
- **sys.certificates**
- **sys.database_principals**

- Первые три представления каталога предоставляют информацию обо всех симметричных ключах, всех асимметричных ключах и всех сертификатах, установленных в текущей базе данных, соответственно. Представление каталога sys.database_principals предоставляет информацию обо всех принципалах в текущей базе данных.
- ***Принципалы (principals)*** - это субъекты, которые имеют разрешение на доступ к определенной сущности. Типичными принципалами являются учетные записи Windows и SQL Server. Кроме этих принципалов, также существуют группы Windows и роли SQL Server. Группа Windows - это коллекция учетных записей и групп Windows. Присвоение учетной записи пользователя членство в группе дает этому пользователю все разрешения, предоставленные данной группе. Подобным образом роль является коллекцией учетных записей.

- Представление каталога sys.database_principals можно соединить с любым из первых трех, чтобы получить информацию о владельце определенного ключа. В примере ниже показано получение информации о существующих сертификатах. Подобным образом можно получить информацию о симметричных и асимметричных ключах.

```sql
USE SampleDb;

SELECT p.name, c.name, certificate_id
    FROM sys.certificates c, sys.database_principals p
    WHERE p.principal_id = p.principal_id
```

Results | Messages

	name	name	certificate...
1	public	cert01	256
2	dbo	cert01	256
3	guest	cert01	256
4	INFORMATION_SCHEMA	cert01	256
5	sys	cert01	256
6	db_owner	cert01	256
7	db_accessadmin	cert01	256
8	db_securityadmin	cert01	256
9	db_ddladmin	cert01	256
10	db_backupoperator	cert01	256
11	db_datareader	cert01	256
12	db_datawriter	cert01	256
13	db_denydatareader	cert01	256
14	db_denydatawriter	cert01	256

Расширенное управление ключами SQL Server

Следующим шагом к обеспечению более высокого уровня безопасности ключей является использование ***расширенного управления ключами (Extensible Key Management - EKM)***. Основными целями расширенного управления ключами являются следующие:

- повышение безопасности ключей посредством выбора поставщика функций шифрования;
- общее управление ключами по всему приложению.

Расширенное управление ключами позволяет сторонним разработчикам регистрировать свои устройства в компоненте Database Engine. Когда такие устройства зарегистрированы, регистрационные имена (logins) в SQL Server могут использовать хранящиеся в них ключи шифрования, а также эффективно использовать продвинутые возможности шифрования, поддерживаемые этими модулями. Расширенное управление ключами позволяет защитить данные от доступа администраторов базы данных (за исключением членов группы sysadmin). Таким образом система защищается от пользователей с повышенными привилегиями. Данные можно зашифровывать и расшифровывать, используя инструкции шифрования языка Transact-SQL, а SQL Server может использовать внешнее устройство расширенного управления ключами для хранения ключей.

Способы шифрования данных

SQL Server поддерживает два способа шифрования данных:

- шифрование на уровне столбцов;
- прозрачное шифрование данных.

- Шифрование на уровне столбцов позволяет шифровать конкретные столбцы данных. Для реализации этого способа шифрования используется несколько пар сопряженных функций. Далее мы не будем рассматривать этот метод шифрования, поскольку его реализация является сложным ручным процессом, требующим внесения изменений в приложение.
- Прозрачное шифрование данных является новой возможностью базы данных, которая зашифровывает файлы базы данных автоматически, не требуя внесения изменений в какие-либо приложения. Таким образом можно предотвратить доступ к информации базы данных несанкционированным лицам, даже если они смогут получить в свое распоряжение основные или резервные файлы базы данных.
- Файлы базы данных зашифровываются на уровне страниц. Страницы зашифрованной базы данных шифруются перед тем, как они записываются на диски и расшифровываются при считывании с диска в память.

Как и большинство других методов шифрования, прозрачное шифрование данных основано на ключе шифрования. В нем используется симметричный ключ, посредством которого и защищается база данных.

Применения прозрачного шифрования для защиты определенной базы данных реализуется в четыре этапа:

- Используя инструкцию CREATE MASTER KEY, создается главный ключ базы данных.
- С помощью инструкции CREATE CERTIFICATE создается сертификат.
- Используя инструкцию **CREATE DATABASE ENCRYPTION KEY**, создается ключ шифрования.
- Выполняется конфигурирование базы данных для использования шифрования. (Этот шаг можно реализовать, присвоив параметру ENCRYPTION инструкции ALTER DATABASE значение ON.)