Харківський національний університет імені В.Н. Каразіна

Лекция

«Шифрование и хеширование данных в базах данных»

1. Шифрование данных в СУБД Oracle.
2. Управление ключами в СУБД Oracle.
3. Прозрачное шифрование данных в СУБД Oracle.
4. Криптографическое хеширование.

Шифрование данных в СУБД Oracle

При использовании подпрограмм шифрования и алгоритма MD5 пакета **DBMS_OBFUSCATION_TOOLKIT** должны выполняться следующие требования:

- длина шифруемых данных должна быть кратна 8 (так 9-байтовое поле типа VARCHAR2 надо будет дополнять до 16 байт);
- ключ, используемый для шифрования данных, должен иметь длину 8 байт для процедуры **DESEncrypt** и 16 или 24 байт для процедур **DES3Decrypt**;
- в зависимости от используемого вида шифрования необходимо вызывать разные подпрограммы (при 56-битовом шифровании вызываются подпрограммы **DESENCRYPT** и **DESDECRYPT**, при 112/168-битовом –**DES3ENCRYPT** и **DES3DECRYPT**;
- подпрограммы шифрования в версии Oracle 8.1.6 являются процедурами, поэтому их нельзя использовать в SQL-операторах (процедуры нельзя вызывать в SQL-операторах непосредственно);
- стандартные подпрограммы шифрования позволяют непосредственно шифровать данные объемом до 32 Кбайт;
- среди подпрограмм шифрования в версии Oracle 8.1.7 есть функции. Однако эти функции перегружены так, что использовать их в SQL-операторах тоже нельзя;
- подпрограммы, реализующие алгоритм MD5, перегружены аналогично и тоже не могут использоваться в SQL-операторах.

Шифрование данных в СУБД Oracle

Пример 1. Зашифрование строки «*SHHH..TOP SECRET*»:

```
SQL> set serverout on
SQL> DECLARE
  2  l_enc_val    VARCHAR2 (200);
  3  BEGIN
  4  DBMS_OBFUSCATION_TOOLKIT.des3encrypt
  5  (input_string          => 'SHHH..TOP SECRET',
  6  key_string             => 'ABCDEFGHIJKLMNOP',
  7  encrypted_string       => l_enc_val
  8  );
  9  DBMS_OUTPUT.put_line ('Encrypted Value = ' || l_enc_val);
 10  END;
 11  /
Encrypted Value = ийVжюTF.(e)♥ьо╠0

PL/SQL procedure successfully completed.
```

Шифрование данных в СУБД Oracle

Модификация примера 1:

```
SQL> DECLARE
  2     l_enc_val   VARCHAR2 (200);
  3  BEGIN
  4  DBMS_OBFUSCATION_TOOLKIT.des3encrypt
  5  (input_string     => 'SHHH..TOP SECRET',
  6   key_string       => 'ABCDEFGHIJKLMNOP',
  7   encrypted_string => l_enc_val);
  8   l_enc_val := RAWTOHEX (UTL_RAW.cast_to_raw (l_enc_val));
  9   DBMS_OUTPUT.put_line ('Encrypted Value = ' ||
l_enc_val);
 10  END;
 11  /

Encrypted Value = A86A56A6EE92462E28652903ECAEC730

PL/SQL procedure successfully completed.
```

Шифрование данных в СУБД Oracle

Основываясь на программах шифрования из пакета **DBMS_OBFUSCATION_TOOLKIT**, составим несколько функций, чтобы сделать процесс шифрования более простым и гибким.

```
CREATE OR REPLACE FUNCTION get_enc_val
(p_in_val IN VARCHAR2, p_key IN VARCHAR2)
RETURN VARCHAR2
IS
l_enc_val   VARCHAR2 (200);
BEGIN
l_enc_val := DBMS_OBFUSCATION_TOOLKIT.des3encrypt
(input_string      => p_in_val,
 key_string        => p_key);
l_enc_val := RAWTOHEX (UTL_RAW.cast_to_raw (l_enc_val));
RETURN l_enc_val;
END;
```

Данную функцию шифрования вы можете использовать разными способами – для вставки данных в зашифрованные столбцы, передачи зашифрованных данных в другие процедуры и функции и т. п.

Шифрование данных в СУБД Oracle

Протестируем данную функцию на разных входных значениях.
Пример 1.

```
SQL> set serverout on
SQL> DECLARE
  2  v_enc   VARCHAR2 (200);
  3  BEGIN
  4   v_enc := get_enc_val ('SHHH..TOP SECRET', 'ABCDEFGHIJKLMNOP');
  5   DBMS_OUTPUT.put_line ('Encrypted value = ' || v_enc);
  6  END;
  7  /
```

Encrypted value = A86A56A6EE92462E28652903ECAEC730

PL/SQL procedure successfully completed.

Шифрование данных в СУБД Oracle

В первом примере мы зашифровывали строку «*SHHH..TOP SECRET*». Теперь зашифруем другое значение:

```
SQL> DECLARE
  2   v_enc    VARCHAR2 (200);
  3   BEGIN
  4   v_enc := get_enc_val ('The DIFFERENT VALUE', 'ABCDEFGHIJKLMNOP');
  5   DBMS_OUTPUT.put_line ('Encrypted value = ' || v_enc);
  6   END;
  7   /
DECLARE
*
ERROR at line 1:
ORA-28232: invalid input length for obfuscation toolkit
ORA-06512: at "SYS.DBMS_OBFUSCATION_TOOLKIT_FFI", line 59
ORA-06512: at "SYS.DBMS_OBFUSCATION_TOOLKIT", line 216
ORA-06512: at "SCOTT.GET_ENC_VAL", line 7
ORA-06512: at line 4
```

Шифрование данных в СУБД Oracle

Изменения в функции :

```
SQL> CREATE OR REPLACE FUNCTION get_enc_val
  2  (p_in_val IN VARCHAR2, p_key IN VARCHAR2)
  3  RETURN VARCHAR2
  4  IS
  5     l_enc_val   VARCHAR2 (200);
  6     l_in_val    VARCHAR2 (200);
  7  BEGIN
  8  l_in_val := RPAD (p_in_val, (8 * ROUND (LENGTH (p_in_val) / 8, 0) + 8));
  9  l_enc_val :=
 10  DBMS_OBFUSCATION_TOOLKIT.des3encrypt
 11  (input_string     => l_in_val,
 12   key_string       => p_key);
 13  l_enc_val := RAWTOHEX (UTL_RAW.cast_to_raw (l_enc_val));
 14  RETURN l_enc_val;
 15  END;
 16  /

Function created.
```

Шифрование данных в СУБД Oracle

Измененная функция шифрования:

```
SQL> CREATE OR REPLACE FUNCTION get_enc_val (
  2     p_in_val    IN   VARCHAR2,
  3     p_key       IN   VARCHAR2,
  4     p_iv        IN   VARCHAR2 := NULL
  5  )
  6     RETURN VARCHAR2
  7  IS
  8     l_enc_val   VARCHAR2 (200);
  9     l_in_val    VARCHAR2 (200);
 10     l_iv        VARCHAR2 (200);
 11  BEGIN
 12   l_in_val := RPAD (p_in_val, (8 * ROUND (LENGTH (p_in_val) / 8, 0) + 8));
 13     l_iv := RPAD (p_iv, (8 * ROUND (LENGTH (p_iv) / 8, 0) + 8));
 14     l_enc_val :=
 15        DBMS_OBFUSCATION_TOOLKIT.des3encrypt
 16        (input_string    => l_in_val,
 17         key_string      => p_key,
 18         iv_string       => l_iv);
 19     l_enc_val := RAWTOHEX (UTL_RAW.cast_to_raw (l_enc_val));
 20     RETURN l_enc_val;
 21  END;
 22  /
```

Шифрование данных в СУБД Oracle

Расшифровывание данных

```
SQL> DECLARE
  2     l_enc_val    VARCHAR2 (2000);
  3     l_dec_val    VARCHAR2 (2000) := 'Clear Text Data';
  4     l_key        VARCHAR2 (2000) := 'ABCDEFGHIJKLMNOP';
  5  BEGIN
  6  l_enc_val := get_enc_val (l_dec_val, l_key, '12345678');
  7  l_dec_val :=
  8  DBMS_OBFUSCATION_TOOLKIT.des3decrypt
  9  (input_string  => UTL_RAW.cast_to_varchar2 (HEXTORAW (l_enc_val)),
 10   key_string    => l_key);
 11  DBMS_OUTPUT.put_line ('Decrypted Value = ' || l_dec_val);
 12  END;
 13  /

Decrypted Value = s}_2OS®Ixt Data

PL/SQL procedure successfully completed.
```

Шифрование данных в СУБД Oracle

Расшифровывание данных **исправление ошибки:**

```
SQL> DECLARE
  2     l_enc_val    VARCHAR2 (2000);
  3     l_dec_val    VARCHAR2 (2000) := 'Clear Text Data';
  4     l_key        VARCHAR2 (2000) := 'ABCDEFGHIJKLMNOP';
  5  BEGIN
  6   l_enc_val := get_enc_val (l_dec_val, l_key, '12345678');
  7   l_dec_val :=
  8   DBMS_OBFUSCATION_TOOLKIT.des3decrypt
  9   (input_string   => UTL_RAW.cast_to_varchar2 (HEXTORAW (l_enc_val)),
 10    key_string    => l_key,
 11    iv_string     => '12345678'
 12              );
 13     DBMS_OUTPUT.put_line ('Decrypted Value = ' || l_dec_val);
 14  END;
 15  /

Decrypted Value = Clear Text Data

PL/SQL procedure successfully completed.
```

Шифрование данных в СУБД Oracle

***Шифрование данных типа* RAW**

Когда следует использовать шифрование в формате RAW?

- *Во-первых*, шифрование в формате RAW используется для данных типа BLOB.
- *Во-вторых*, шифрование в формате RAW используется в случае, если в базе данных используются буквы не английского алфавита. Если вы пользуетесь функциональностью *Oracle Globalization Support* (ранее *National Language Support – NLS*), то шифрование в формате RAW обеспечит обработку таких символов без выполнения каких-либо дополнительных операций (в частности, при экспорте и импорте данных). Зашифрованные данные можно будет перемещать из одной базы данных в другую, не опасаясь их повреждения.

Однако в общем случае по возможности избегайте манипуляций с типом RAW, *если изначально известно, что ваши данные имеют символьный тип, и вы используете только один набор символов.*

Шифрование данных в СУБД Oracle

Итак, основные правила шифрования в младших версиях Oracle, использующих пакет DBMS_OBFUSCATION_TOOLKIT:

- Шифрование может выполняться по алгоритму DES или DES3 (*Triple DES*), причем DES3 является предпочтительным.
- Шифрование DES3 может быть двух- или трехпроходным. По умолчанию используются два прохода.
- Длина шифруемой строки должна быть кратна восьми.
- Ключ, используемый для зашифровывания, должен использоваться и при расшифровывании.
- Длина ключа должна быть не меньше 16 для DES и двухпроходного DES3, и не меньше 24 для трехпроходного DES3.
- Для усиления защиты данных возможно использование вектора инициализации. Если вектор инициализации использован при зашифровывании, его также необходимо указать при расшифровывании.
- Так как вектор инициализации присоединяется в начало входного значения, длина полученной объединенной строки должна быть кратна восьми.

Замечание. Невозможно зашифровать данные, уже зашифрованные средствами пакета DBMS_OBFUSCATION_TOOLKIT. При подобной попытке пакет генерирует ошибку ORA-28233 в связи с невозможностью двойного шифрования.

Шифрование данных в СУБД Oracle

При выборе ключа шифрования следует помнить:

- чем длиннее ключ, тем сложнее его угадать (двухпроходный метод допускает использование ключа из 128 бит, а трехпроходный – из 192 бит; следует выбирать наиболее длинный из возможных ключей);
- ключ должен быть не только длинным, он не должен соответствовать никакому шаблону, который было бы легко угадать.

В пакет DBMS_OBFUSCATION_TOOLKIT включены две функции DESGETKEY и DES3GETKEY (а также процедура им соответствующие, причем все они перегружены с несколькими типами данных: RAW и VARCHAR2), которые позволяют сгенерировать криптографически допустимый ключ.

Пример вызова функции :

```
l_ret := DBMS_OBFUSCATION_TOOLKIT.desgetkey
(seed_string => l_seed);
```

Значение переменной l_seed – случайная строка длиной 80 символов (более длинное значение будет принято, но использованы будут только 80 символов). Возвращаемое значение записывается в переменную l_ret.

Шифрование данных в СУБД Oracle

Пакет **DBMS_CRYPTO** имеет ряд преимуществ перед **DBMS_OBFUSCATION_TOOLKIT**:

- ❑ больший выбор алгоритмов шифрования, в частности, поддержка последнего стандарта AES (*Advanced Encryption Standard*);
- ❑ возможность поточного шифрования;
- ❑ поддержка алгоритма SHA-1 (*Secure Hash Algorithm 1*).
- ❑ способность создания кода MAC;
- ❑ шифрование больших объектов (LOB) в их собственном формате.

Пакет DBMS_CRYPTO, доступный в Oracle 10g, содержит функцию GETRANDOMBYTES, которая может использоваться для формирования криптографически случайных ключей.

В дополнение к генерированию ключей в формате RAW (посредством функции RANDOMBYTES) пакет DBMS_CRYPTO обеспечивает формирование числовых значений, а также двоичных целых. Функция RANDOMINTEGER генерирует двоичный целый ключ, например:

l_ret := DBMS_CRYPTO.randominteger;.

Функция RANDOMNUMBER генерирует ключ целочисленного типа длиной 2^{128}:

l_ret := DBMS_CRYPTO.randomnumber;.

Шифрование данных в СУБД Oracle

Зашифрование данных

После того как ключ готов, приступим собственно к шифрованию данных. Воспользуемся для этого программой ENCRYPT пакета DBMS_CRYPTO.

ENCRYPT перегружена и существует как в виде функции, так и процедуры. В отличие от пакета DBMS_OBFUSCATION_TOOLKIT, такая перегрузка для нее в пакете DBMS_CRYPTO обоснована: функция принимает в качестве входного значения только тип данных RAW, в то время как процедура принимает на вход только значения типов CLOB и BLOB.

Пример интерфейса функции шифрования ENCRYPT, возвращающей значения типа RAW:

```
DBMS_CRYPTO.encrypt(
   src IN RAW,
   typ IN PLS_INTEGER,
   key IN RAW,
   iv  IN RAW           DEFAULT NULL)
 RETURN RAW;
```

где src – входное значение, которое подлежит шифрованию;

key – ключ шифрования;

iv – вектор инициализации;

typ – тип шифрования.

Шифрование данных в СУБД Oracle

Пакеты DBMS_OBFUSCATION_TOOLKIT и DBMS_CRYPTO отличаются способами поддержки различных типов шифрования. Пакет DBMS_OBFUSCATION_TOOLKIT предлагает специальные функции (и соответствующие процедуры) для каждого алгоритма, например DESENCRYPT для DES и DES3ENCRYPT для Triple DES. Пакет DBMS_CRYPTO предоставляет всего одну функцию, а тип шифрования указывается в параметре. Поддерживаемые алгоритмы шифрования и соответствующие им константы приведены в таблице 1.

Таблица 1. – Типы шифрования в пакете DBMS_CRYPTO

Константа	*Описание*	*Реальная длина ключа*
ENCRYPT_DES	*Data Encryption Standard* (DES).	56
ENCRYPT_3DES_2KEY	*Modified Triple Data Encryption Standard* (3DES); обрабатывает каждый блок трижды, используя 2 ключа.	112
ENCRYPT_3DES	*Triple Data Encryption Standard* (3DES); обрабатывает каждый блок трижды.	156
ENCRYPT_AES128	*Advanced Encryption Standard* (AES).	128
ENCRYPT_AES192	*Advanced Encryption Standard* (AES).	192
ENCRYPT_AES256	*Advanced Encryption Standard* (AES).	256
ENCRYPT_RC4	Потоковое шифрование.	

Шифрование данных в СУБД Oracle

Режимы шифрования

При шифровании данных каждый шифруемый блок может быть зашифрован в зависимости от режима шифрования (*ECB*, *CBC*, *CFB*, *OFB*). Чтобы выбрать интересующий вас режим шифрования, укажите соответствующую константу из таблицы 2 в значении параметра typ, например, DBMS_CRYPTO.CHAIN_OFB.

Таблица 2. – Режимы шифрования DBMS_CRYPTO

Константа	*Описание*
CHAIN_CBC	Сцепление блоков шифртекста – *Cipher Block Chaining – CBC.*
CHAIN_ECB	Электронная книга кодов – *Electronic Code Book – ECB.*
CHAIN_CFB	Шифрование с обратной связью по шифртексту – *Cipher Feedback – CFB.*
CHAIN_OFB	Шифрование с обратной связью по выходу – *Output Feedback – OFB.*

Шифрование данных в СУБД Oracle

Типы дополнения

При использовании пакета DBMS_OBFUSCATION_TOOLKIT необходимо явно дополнить данные так, чтобы их длина была кратна размеру блока. Однако этот подход не является криптографически надежным. DBMS_CRYPTO позволяет указать необходимый тип дополнения («набивку» – см. таблицу 3). Большинство компаний для дополнения использует метод PKCS#5 – для этого требуется указать соответствующую константу (PAD_PKCS5) из таблицы 3 в значении параметра typ – DBMS_CRYPTO.PAD_PKCS5.

Таблица 3. – Типы дополнения DBMS_CRYPTO

Константа	*Описание*
PAD_PKCS5	Дополнение средствами криптографической системы с общим ключом (*Public Key Cryptography System* #5).
PAD_ZERO	Дополнение нулями.
PAD_NONE	Отсутствие дополнения. Используется при уверенности в том, что длина данных уже кратна размеру шифруемого блока (кратно 8).

Шифрование данных в СУБД Oracle

***Объединение опций в параметре* typ**

Предположим, что вы выбрали следующие опции шифрования:
метод дополнения: дополнение нулями (PAD_ZERO);
алгоритм шифрования: 128-битный ключ, алгоритм *AES* (ENCRYPT_AES128);
режим шифрования: блочное шифрование с обратной связью по шифртексту *Cipher Feedback* (CHAIN_CFB);
Объединяем эти опции в значении параметра typ следующим образом :

```
typ => DBMS_CRYPTO.pad_zero + DBMS_CRYPTO.encrypt_aes128
       + DBMS_CRYPTO.chain_cfb
```

Аналогично можно задавать любую комбинацию опций функции ENCRYPT.
Рассмотрим пример полного вызова функции:

```
DECLARE
   l_enc   RAW(2000);
   l_in    RAW(2000);
   l_key   RAW(2000);
BEGIN
   l_enc :=
      DBMS_CRYPTO.encrypt (src      => l_in,
                           KEY      => l_key,
                           typ      =>   DBMS_CRYPTO.pad_zero
                                       + DBMS_CRYPTO.encrypt_aes128
                                       + DBMS_CRYPTO.chain_cfb
                          );
END;
```

Шифрование данных в СУБД Oracle

Для удобства работы пакет предлагает две константы с предопределенными комбинациями значений для опций шифрования, режима и дополнения (таблица 4).

Таблица 4. – Константы пакета DBMS_CRYPTO *для параметра* typ

Константа	*Шифрование*	*Дополнение*	*Режим*
DES_CBC_PKCS5	ENCRYPT_DES	PAD_PKCS5	CHAIN_CBC
DES3_CBC_PKCS5	ENCRYPT_3DES	PAD_PKCS5	CHAIN_CBC

Если надо использовать алгоритм шифрования DES, дополнение по системе PKCS#5 и режим шифрования CBC, то следует задать константы следующим образом:

```
DECLARE
   l_enc   RAW(2000);
   l_in    RAW(2000);
   l_key   RAW(2000);
BEGIN
   l_enc :=
 DBMS_CRYPTO.encrypt (src      => l_in,
                      KEY      => l_key,
                      typ      => DBMS_CRYPTO.des_cbc_pkcs5
                      );
END;
```

Шифрование данных в СУБД Oracle

Для Oracle Database 12c

Package Feature	DBMS_CRYPTO
Cryptographic algorithms	DES, 3DES, AES, RC4, 3DES_2KEY
Padding forms	PKCS5, zeroes
Block cipher chaining modes	CBC, CFB, ECB, OFB
Cryptographic hash algorithms	MD5, SHA-1, SHA-2 (SHA-256, SHA-384, SHA-512), MD4
Keyed hash (MAC) algorithms	HMAC_MD5, HMAC_SH1, HMAC_SH256, HMAC_SH384, HMAC_SH512
Cryptographic pseudo-random number generator	`RAW`, `NUMBER`, `BINARY_INTEGER`
Database types	`RAW`, `CLOB`, `BLOB`

Шифрование данных в СУБД Oracle

Функция ENCRYPT принимает входные значения в формате RAW, так что исходные данные придется преобразовывать к типу RAW. Сделаем следующее:

```
l_in := UTL_I18N.string_to_raw (p_in_val, 'AL32UTF8');
```

Ранее использовался встроенный пакет UTL_RAW для преобразования значений типа VARCHAR в RAW. В данном случае используется для такого преобразования функцию UTL_I18N.
Почему STRING_TO_RAW, а не UTL_RAW.CAST_TO_RAW ?

Функция ENCRYPT принимает на вход значения типа RAW и, кроме того, требует использования специального набора символов AL32UTF8, который не обязательно является набором символов базы данных. Так что фактически необходимо выполнить два преобразования:
из текущего набора символов базы данных в набор символов AL32UTF8;
из VARCHAR2 в RAW.

Функция CAST_TO_RAW не умеет выполнять преобразование набора символов, а функция STRING_TO_RAW встроенного пакета UTL_i18n может выполнить оба преобразования.

Шифрование данных в СУБД Oracle

Расшифровывание данных

Обратная задача – процесс расшифровывания, при котором зашифрованная строка расшифровывается с помощью того же ключа, который был использован для шифрования.

При расшифровывании зашифрованного значения необходимо использовать те же ключ, алгоритм шифрования, тип дополнения и режим шифрования, что и при зашифровывании.

Управление ключами в СУБД Oracle

Существует три основных подхода к управлению ключами:

1)использование одного и того же ключа для всей базы данных;

2)использование различных ключей для разных строк таблиц БД;

3)комбинированный подход.

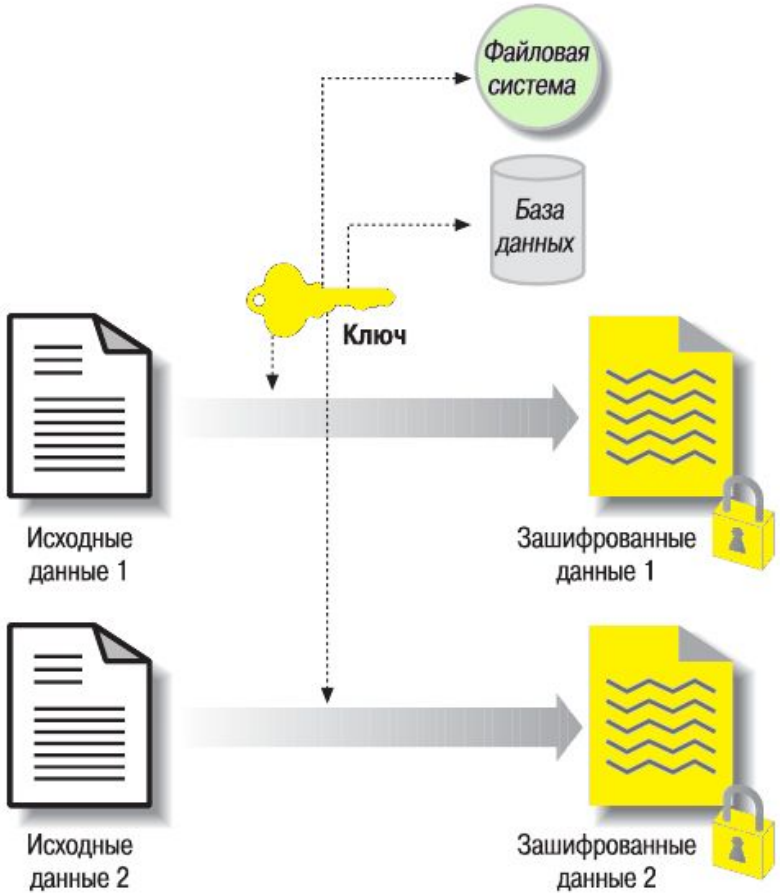


Рис. 1 – Использование одного ключа для всей базы данных

Управление ключами в СУБД Oracle

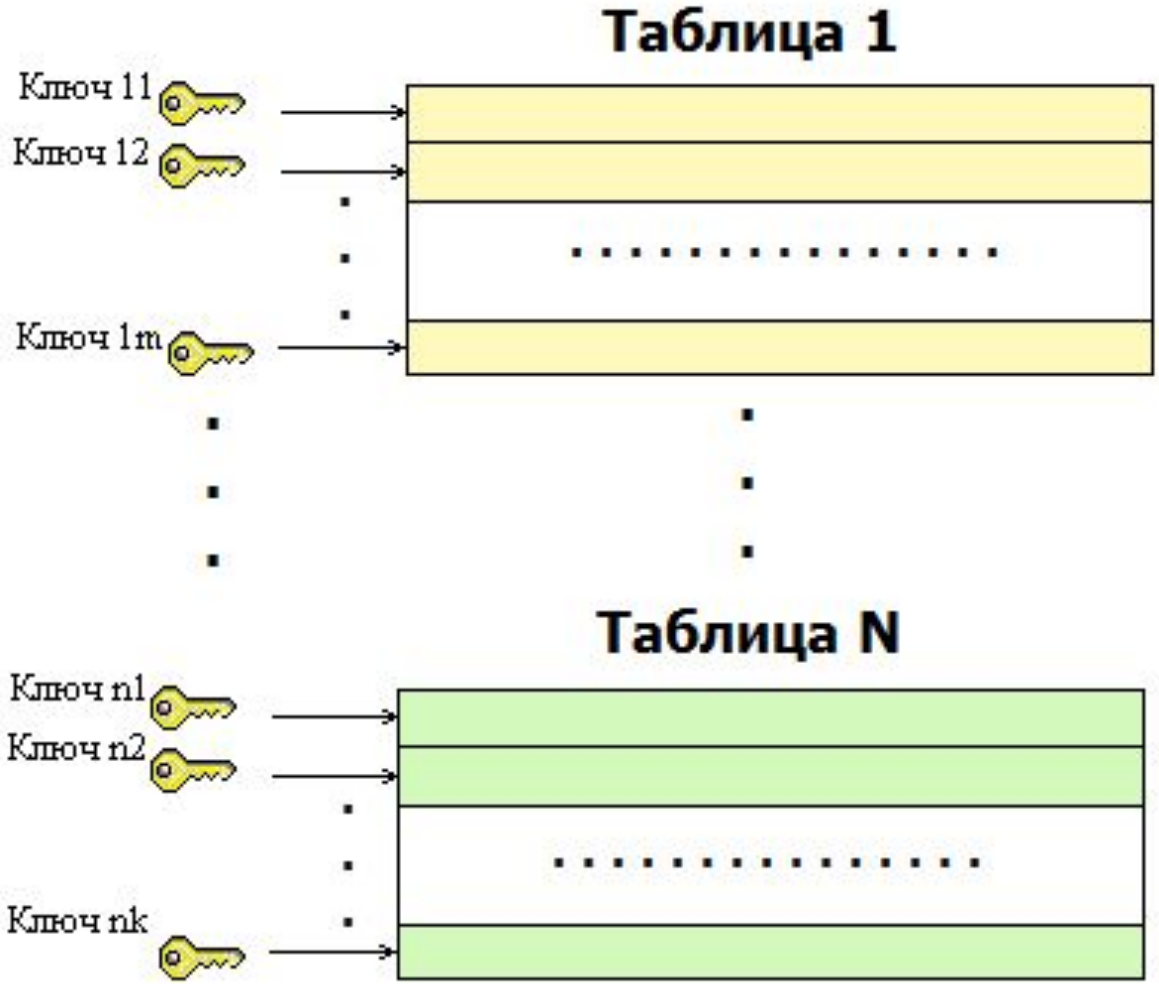


Рис. 2 – Использование нового ключа для каждой строки

Управление ключами в СУБД Oracle

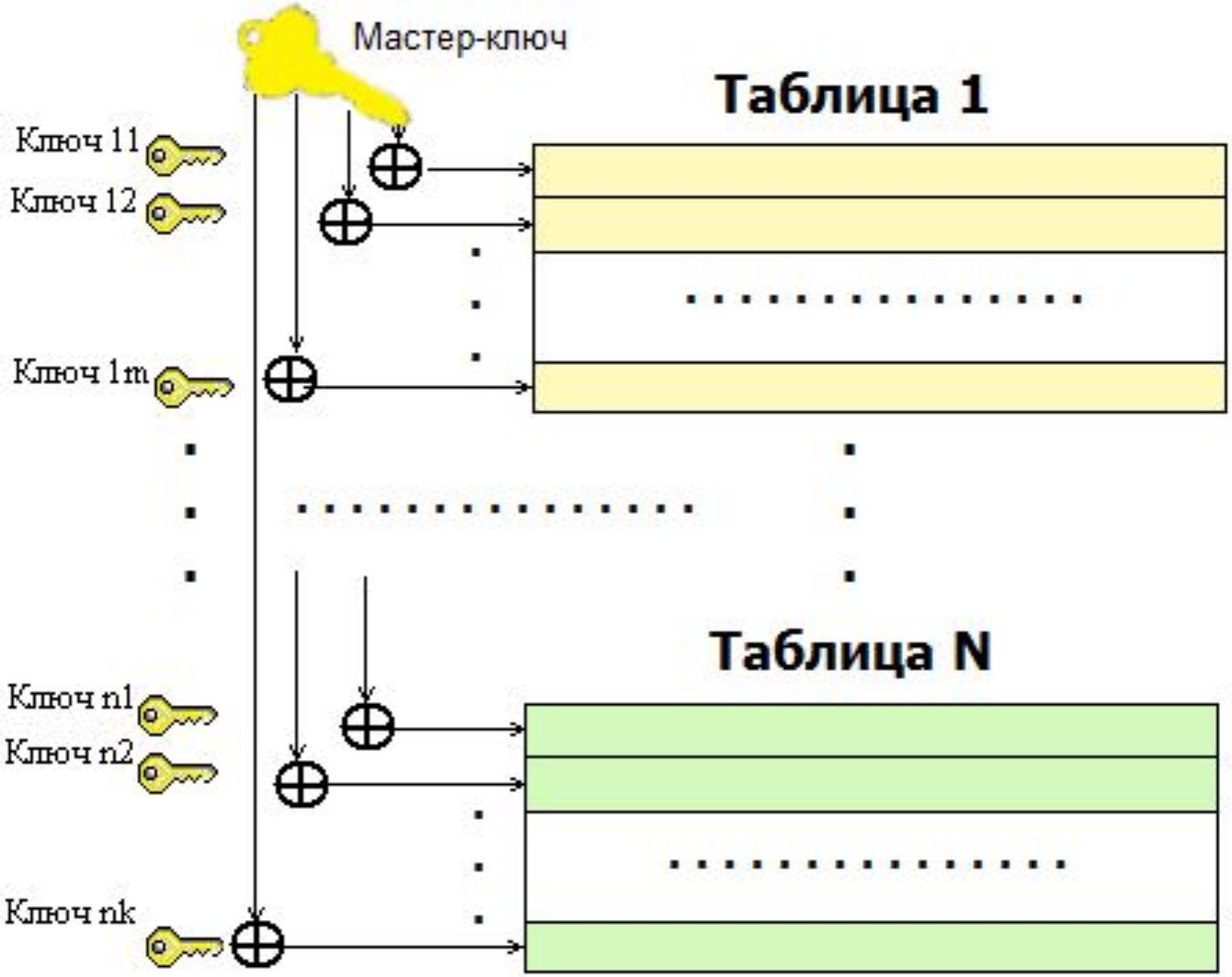


Рис. 3 – Использование мастер-ключа

Управление ключами в СУБД Oracle

```
SQL> REM Определяем переменную для хранения зашифрованного значения
SQL> VARIABLE enc_val varchar2(2000);
SQL> DECLARE
  2  l_key          VARCHAR2 (2000) := '0123456789012345';
  3  l_master_key   VARCHAR2 (2000) := '&master_key';
  4  l_in_val       VARCHAR2 (2000) := 'Секретные данные';
  5  l_mod          NUMBER
  6                    :=   DBMS_CRYPTO.encrypt_aes128
  7                       + DBMS_CRYPTO.chain_cbc
  8                       + DBMS_CRYPTO.pad_pkcs5;
  9  l_enc          RAW (2000);
 10  l_enc_key      RAW (2000);
 11  BEGIN
 12  l_enc_key :=
 13        UTL_RAW.bit_xor (utl_i18n.string_to_raw (l_key, 'AL32UTF8'),
 14                         utl_i18n.string_to_raw (l_master_key, 'AL32UTF8')
 15                         );
 16  l_enc :=
 17  DBMS_CRYPTO.encrypt (utl_i18n.string_to_raw (l_in_val, 'AL32UTF8'),
 18                       l_mod,
 19                       l_enc_key
 20                       );
 21  DBMS_OUTPUT.put_line ('Encrypted=' || l_enc);
 22  :enc_val := RAWTOHEX (l_enc);
 23  END;
 24  /
Enter value for master_key: Master-Key012345
old   3: l_master_key   VARCHAR2 (2000) := '&master_key';
new   3: l_master_key   VARCHAR2 (2000) := 'Master-Key012345';
Encrypted=299E749638D5B64E1FED590AFABA5439B313DD0659D44BABFACA2F66804753B38AEA9A
58162B2E9309031BD22A258A83
```

Прозрачное шифрование данных в СУБД Oracle

Если вы храните ключ шифрования и зашифрованные данные в базе данных, то возникает еще одна потенциальная угроза безопасности. При краже дисков, на которых хранится вся база данных, все данные сразу же рассекречиваются.

Для того чтобы обойти эту проблему, следует хранить ключ отдельно, вне диска, на котором хранятся зашифрованные этим ключом данные. СУБД Oracle позволяет раздельно хранить зашифрованные данные и секретный ключ посредством применения, так называемого прозрачного шифрования (*Transparent Data Encryption – TDE*). Почему прозрачного, потому что данная функциональная возможность не требует какого-либо изменения кода приложения.

Суть прозрачного шифрования состоит в том, что используется сочетание двух ключей (рис. 4): *ключа для каждой таблицы базы данных*, который является уникальным; *мастер-ключа*, который хранится вне базы данных в бумажнике – Oracle Wallet.

Нельзя использовать прозрачное шифрование в таблицах, принадлежащих пользователю SYS.

Прозрачное шифрование данных в СУБД Oracle

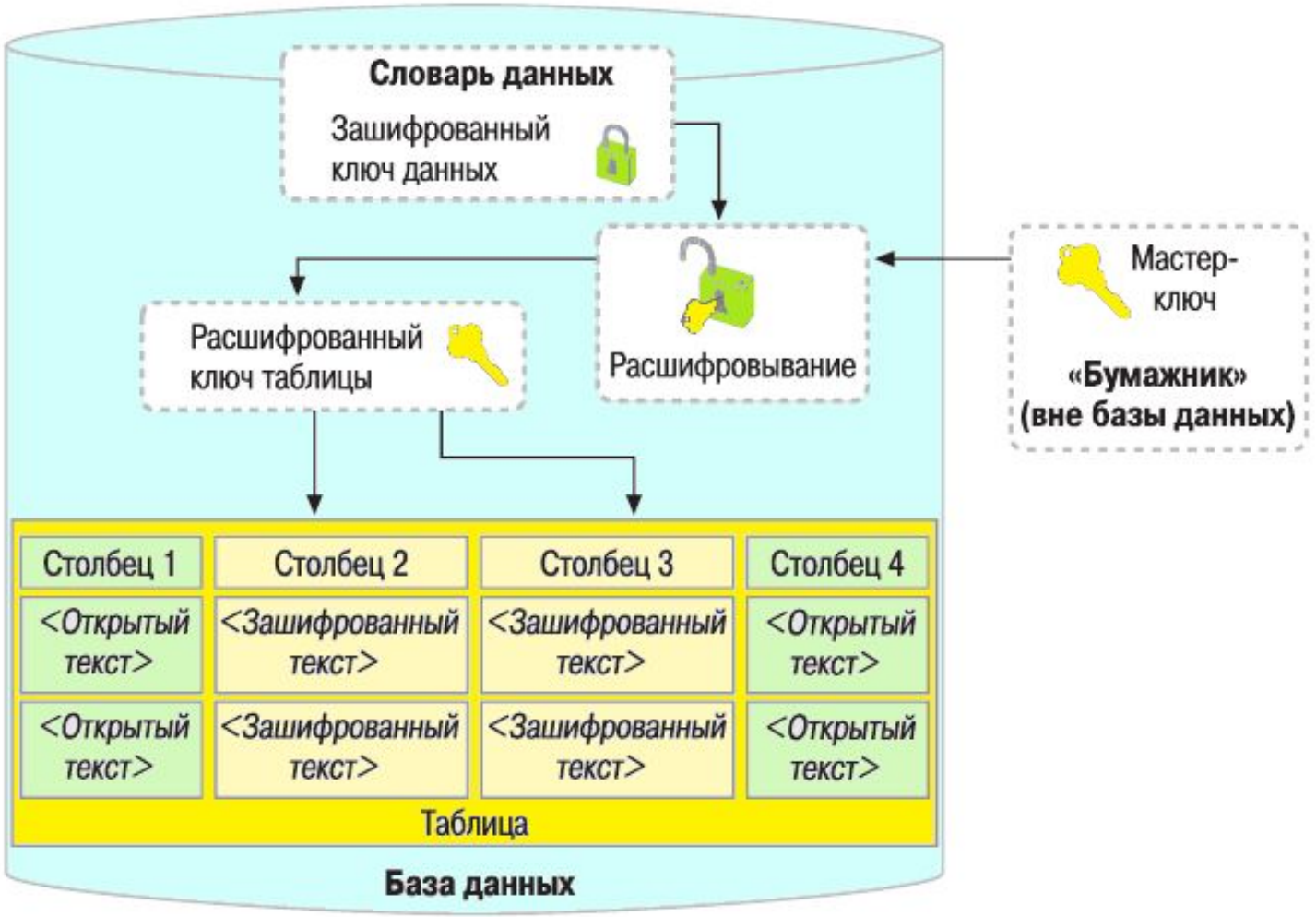


Рис. 4 – Модель прозрачного шифрования данных

Прозрачное шифрование данных в СУБД Oracle

Процесс управления «бумажниками»

1. *Выбор местоположения «бумажника»*.

Перед первым применением TDE необходимо создать «бумажник», в котором будет храниться мастер-ключ. По умолчанию «бумажник» создается в каталоге $ORACLE_BASE/admin/$ORACLE_SID/wallet. Вы можете выбрать другой каталог, указав его в файле SQLNET.ORA. Например, если вы хотите, чтобы «бумажник» хранился в каталоге /wallet, добавьте приведенные ниже строки в файл SQLNET.ORA.

Для определенности будем полагать, что выбран каталог по умолчанию.

```
ENCRYPTION_WALLET_LOCATION =
  (SOURCE=
     (METHOD=file)
     (METHOD_DATA=
       (DIRECTORY=/wallet)))
```

Прозрачное шифрование данных в СУБД Oracle

2. Установка пароля для «бумажника».

Для этого сначала подключаемся к базе данных с правами SYS. Затем создаем «бумажник» и указываем пароль для доступа к нему. Для чего выполняем следующий оператор:

```
ALTER SYSTEM SET ENCRYPTION KEY IDENTIFIED BY "password-wallet";
```

Если возникли некоторые проблемы с выполнением данного оператора, например, выдается ошибка:

```
ORA-28368: cannot auto-create wallet
```

попробуйте завершить сеанс и затем подключиться вновь с правами SYS.

Данный оператор выполняет три действия:

- создает «бумажник» в каталоге, определенном в шаге 1 (бумажник Oracle Wallet представляет собой двоичный файл ewallet.p12, составленный по промышленному стандарту PKCS #12);
- устанавливает для «бумажника» пароль – «`password-wallet`»;
- открывает «бумажник» для сохранения и извлечения ключей средствами TDE.

Пароль является регистрозависимым и должен заключаться в двойные кавычки.

Прозрачное шифрование данных в СУБД Oracle

3. Открытие «бумажника».

На предыдущем шаге бумажник был открыт для работы с ним.

Однако после того как бумажник создан, вам не придется пересоздавать его. После запуска базы данных вам нужно будет только открыть «бумажник», используя установленный пароль:

```
ALTER SYSTEM SET ENCRYPTION WALLET OPEN IDENTIFIED BY "password-wallet";
```

или

```
ALTER SYSTEM SET ENCRYPTION WALLET OPEN AUTHENTICATED BY "password-wallet";
```

Вы можете закрыть «бумажник» следующей командой:

```
ALTER SYSTEM SET ENCRYPTION WALLET CLOSE;
```

Открытие «бумажника» необходимо для работы средств TDE.

Если «бумажник» не открыт, то все незашифрованные столбцы будут доступны, а зашифрованные столбцы – недоступны.

Прозрачное шифрование данных в СУБД Oracle

Для того чтобы воспользоваться преимуществами TDE, добавьте предложение ENCRYPT (доступное только в Oracle 10g Release 2) для каждого шифруемого столбца в оператор создания вашей таблицы:

```
CREATE TABLE accounts
(
   acc_no        NUMBER         NOT NULL,
   first_name    VARCHAR2(30)   NOT NULL,
   last_name     VARCHAR2(30)   NOT NULL,
   SSN           VARCHAR2(9)                ENCRYPT USING 'AES128',
   acc_type      VARCHAR2(1)    NOT NULL,
   folio_id      NUMBER                     ENCRYPT USING 'AES128',
   sub_acc_type VARCHAR2(30),
   acc_open_dt   DATE           NOT NULL,
   acc_mod_dt    DATE,
   acc_mgr_id    NUMBER
);
```

Прозрачное шифрование данных в СУБД Oracle

Использование TDE для уже существующих таблиц

Выше было показано, как можно использовать TDE при создании новой таблицы. Так же можно зашифровать и столбец существующей таблицы. Для шифрования столбца LAST_NAME таблицы ACCOUNTS используем такой оператор:

```
ALTER TABLE accounts MODIFY (LAST_NAME ENCRYPT);
```

Данный оператор выполняет два действия: создает ключ для столбца SSN, преобразует все значения столбца в зашифрованный формат. Шифрование выполняется внутри базы данных. По умолчанию (вначале) используется алгоритм AES со 192-битным ключом. Вы можете выбрать другой алгоритм шифрования, указав его название в операторе. Например, чтобы выбрать алгоритм AES с 128-битным ключом, используйте следующий оператор:

```
ALTER TABLE accounts MODIFY (LAST_NAME ENCRYPT USING 'AES128');
```

В качестве параметров также можно было бы указать AES128, AES256 или 3DES168 (для трехпроходного алгоритма DES со 168-битным ключом).

Прозрачное шифрование данных в СУБД Oracle

Посмотрим на таблицу после шифрования столбца:

```
SQL>  DESC scott.accounts;
 Name                                      Null?    Type
 ----------------------------------------- -------- ------------
 ACC_NO                                    NOT NULL NUMBER
 FIRST_NAME                                NOT NULL VARCHAR2(30)
 LAST_NAME                                 NOT NULL VARCHAR2(30) ENCRYPT
 SSN                                                VARCHAR2(9) ENCRYPT
 ACC_TYPE                                  NOT NULL VARCHAR2(1)
 FOLIO_ID                                           NUMBER ENCRYPT
 SUB_ACC_TYPE                                       VARCHAR2(30)
 ACC_OPEN_DT                               NOT NULL DATE
 ACC_MOD_DT                                         DATE
 ACC_MGR_ID                                         NUMBER
```

Замечание. Нельзя указывать для разных столбцов одной таблицы разные алгоритмы шифрования. Так в данном случае нельзя использовать оператор

```
ALTER TABLE accounts MODIFY (LAST_NAME ENCRYPT USING 'AES256');
```

так как столбцы SSN и folio_id используют алгоритм AES128, и что вызовет ошибку:
ORA-28340: a different encryption algorithm has been chosen for the table.

Прозрачное шифрование данных в СУБД Oracle

Для поиска зашифрованных столбцов базы данных используйте новое представление словаря данных DBA_ENCRYPTED_COLUMNS.

```
SQL> COL OWNER FORMAT A10
SQL> COL TABLE_NAME FORMAT A15
SQL> COL COLUMN_NAME FORMAT A15
SQL> COL ENCRYPTION_ALG FORMAT A20
SQL> select * from DBA_ENCRYPTED_COLUMNS;

OWNER      TABLE_NAME      COLUMN_NAME     ENCRYPTION_ALG        SAL
---------- --------------- --------------- -------------------- ---
SCOTT      ACCOUNTS        LAST_NAME       AES 128 bits key     YES
SCOTT      ACCOUNTS        SSN             AES 128 bits key     YES
SCOTT      ACCOUNTS        FOLIO_ID        AES 128 bits key     YES
```

Прозрачное шифрование данных в СУБД Oracle

Управление ключами и паролями для TDE

Что будет, если кто-то каким-то образом узнает ваши TDE ключи?

Можно пересоздать зашифрованные значения, выполнив всего один оператор.

В этом операторе можно также выбрать другой алгоритм шифрования, например AES256:

```
SQL> ALTER TABLE accounts REKEY USING 'aes256';

Table altered.

SQL> select * from DBA_ENCRYPTED_COLUMNS;

OWNER      TABLE_NAME      COLUMN_NAME     ENCRYPTION_ALG        SALT
---------- --------------- --------------- --------------------- ----
SCOTT      ACCOUNTS        LAST_NAME       AES 256 bits key      YES
SCOTT      ACCOUNTS        SSN             AES 256 bits key      YES
SCOTT      ACCOUNTS        FOLIO_ID        AES 256 bits key      YES
```

Прозрачное шифрование данных в СУБД Oracle

Фактически команда

```
ALTER TABLE accounts REKEY USING 'aes256';
```

сменяет не только алгоритм, но и ключ шифрования (также один на всю таблицу). Этот собственный ключ шифрования таблицы тут же сам шифруется мастер-ключом из бумажника и запоминается для нее в БД, в системной таблице ENC$. Оттуда он будет браться каждый раз при зашифровании/расшифровании значений полей строк таблицы (см. рис. 4).

Так, при обращении к зашифрованному полю таблицы СУБД возьмет из ENC$ зашифрованный ключ этой таблицы, расшифрует его мастер-ключом из бумажника, и уже восстановленным ключом таблицы расшифрует значение поля.

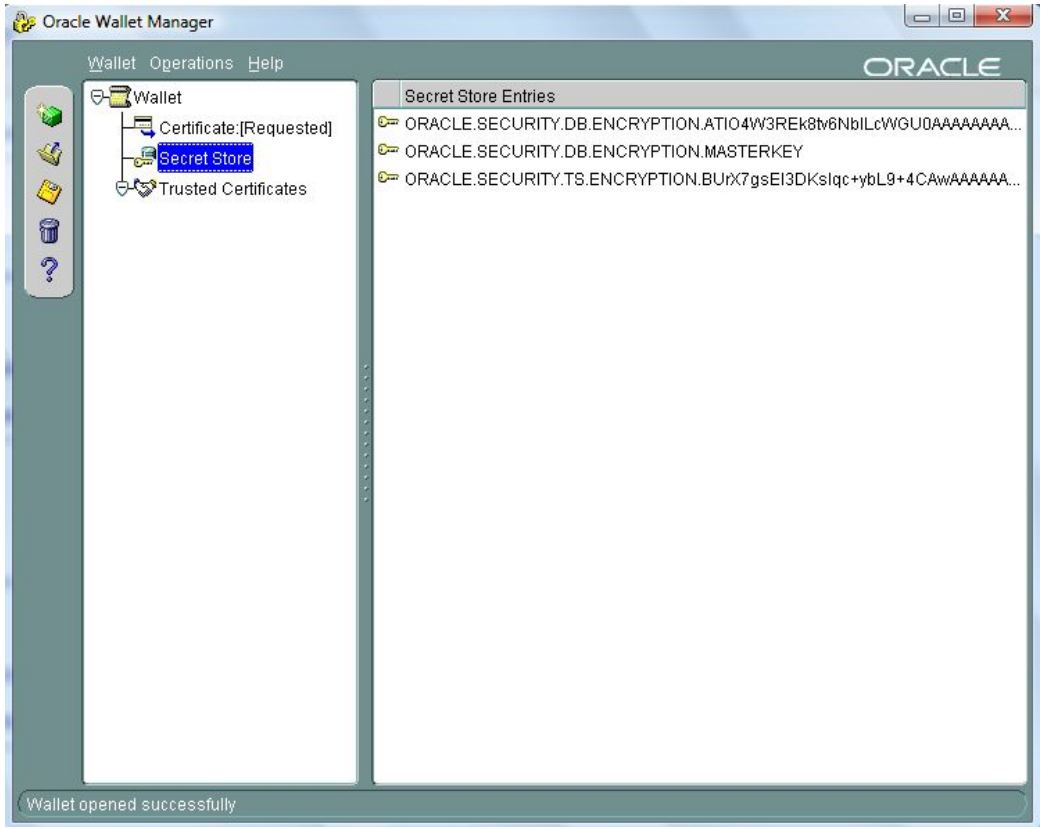
Если нужно сменить хранимые зашифрованные значения, можно просто выполнить следующий оператор:

```
ALTER TABLE closed REKEY;
```

Прозрачное шифрование данных в СУБД Oracle

Если кто-то узнает пароль «бумажника», вы можете изменить его, используя программу Oracle Wallet Manager. Запустив диспетчер «бумажника», выберите в главном меню *Wallet→Open* и укажите место хранения «бумажника» и его пароль. Для изменения пароля выберите *Wallet→Change Password*.

Имейте в виду, что изменение пароля не приводит к изменению ключей.



Прозрачное шифрование данных в СУБД Oracle

Добавим «привязку»

Шифрование предназначено для сокрытия данных, но бывает так, что зашифрованные данные легко угадать из-за повторений в исходных данных. В этом случае зашифрованные значения также будут одинаковыми.

Для предотвращения таких ситуаций к данным добавляется, так называемая «*привязка*» или ее еще называют *соль* (*salt*), благодаря которой зашифрованные значения будут различаться даже для одинаковых исходных данных. TDE использует *соль* по умолчанию (см. запрос к представлению словаря данных

```
select * from DBA_ENCRYPTED_COLUMNS;
```
).

При этом следует помнить, что в некоторых случаях структуры данных могут способствовать повышению производительности базы данных, а добавление «соли» может ее ухудшить.

От применения привязки можно отказаться:

```
ALTER TABLE accounts MODIFY (SSN ENCRYPT NO SALT);
```

Прозрачное шифрование данных в СУБД Oracle

Посмотреть состояние и местонахождение бумажника можно только, начиная с 11 версии Oracle в таблице V$ENCRYPTION_WALLET.

```
SQL> COL WRL_PARAMETER FORMAT A55;
SQL> select * from V$ENCRYPTION_WALLET;

WRL_TYPE   WRL_PARAMETER                                   STATUS
---------- ------------------------------------------------- -------
file       d:\app\HOST-name\admin\ora-sid\wallet           OPEN
```

Замечание. Использование средств TDE невозможно для столбцов, обладающих одним из приведенных ниже свойств:

✔ тип данных BLOB или CLOB;

✔ использование в индексах, отличных от обычных индексов b-tree, таких как bitmap-индексы, индексы на основе функций и т. д.;

✔ использование в ключах секционирования.

Отсутствие возможности использования средств TDE в подобных случаях является еще одной причиной, по которой TDE не может использоваться для всех видов шифрования.

Прозрачное шифрование данных в СУБД Oracle

Создание табличных пространств с зашифрованными данными

Когда возникает надобность шифровать данные многих столбцов во многих таблицах, традиционным способом, рассмотренным выше это делать не всегда удобно и рационально. В 11 версии Oracle для таких случаев имеется обобщенное решение: зашифрование всех данных, помещаемых в конкретное табличное пространство. Такое свойство табличного пространства неизменно и указывается один раз при его создании, например:

```
SQL> CREATE TABLESPACE encrypted_data
  2  DATAFILE 'd:\app\hostname\oradata\orasid\encrf.dat' SIZE 1M
  3  ENCRYPTION USING '3DES168'
  4  DEFAULT STORAGE ( ENCRYPT )
  5  ;

Tablespace created.
```

Оно отображено в поле ENCRYPTED таблицы DBA_TABLESPACES.

Криптографическое хеширование

Отличие хеширования от шифрования в том, что хеширование – это однонаправленный процесс. Вы можете расшифровать зашифрованные данные, но «расхешировать» хеш-значение невозможно. Если вы несколько раз хешируете один и тот же элемент данных, результат будет неизменным при любом количестве выполнений. Если данные каким-то образом меняются, генерируемое хеш-значение изменится, свидетельствуя о «порче» данных.

Хеширование в Oracle 9 i

```
SQL> DECLARE
  2   l_hash      VARCHAR2 (2000);
  3   l_in_val    VARCHAR2 (2000);
  4  BEGIN
  5   l_in_val := 'Account Balance is 12345.67';
  6   l_hash := DBMS_OBFUSCATION_TOOLKIT.MD5 (input_string => l_in_val);
  7   l_hash := RAWTOHEX (UTL_RAW.cast_to_raw (l_hash));
  8   DBMS_OUTPUT.put_line ('Hashed Value = ' || l_hash);
  9  END;
 10  /
Hashed Value = A09308E539C35C97CD612E918BA58B4C
```

Криптографическое хеширование

Как можно заметить хеширование имеет два важных отличия от шифрования:

- при хешировании входную строку не нужно дополнять до определенной длины, как это делается при шифровании;
- при хешировании не используется ключ. А раз ключа нет, то его не нужно хранить или вводить, что делает систему хеширования чрезвычайно простой.

Теоретически возможно получение одного и того же хеш-значения для двух разных входных значений. Однако, используя такие общераспространенные алгоритмы, как MD5 и SHA-1, вы обеспечиваете чрезвычайно малую вероятность хеш-конфликта – порядка $1/10^{38}$ (в зависимости от выбранного алгоритма). Если вы не можете допустить наличия даже столь малой вероятности, то вам придется реализовать логику разрешения конфликтов для хеш-функций.

Криптографическое хеширование

Хеширование в Oracle 10g

На сегодняшний день алгоритм хэширования MD5 считается недостаточно надежным для современной защиты данных и вместо него часто используется алгоритм SHA-1. Но алгоритм SHA-1 не поддерживается в пакете DBMS_OBFUSCATION_TOOLKIT. Он доступен только, начиная с версии Oracle 10g в пакете DBMS_CRYPTO функция HASH. Объявление этой функции:

```
DBMS_CRYPTO.HASH (
   src in raw,
   typ in pls_integer)
 return raw;
```

Функция HASH принимает на вход только значения типа RAW, поэтому необходимо преобразовать входную символьную строку к типу RAW, что реально и делается так же, как ранее для шифрования:

```
l_in := utl_i18n.string_to_raw (p_in_val, 'AL32UTF8');
```

Криптографическое хеширование

Таблица 5. – Алгоритмы хеширования пакета DBMS_CRYPTO

Константа	*Описание*
DBMS_CRYPTO.HASH_MD5	Message Digest 5 (MD5)
DBMS_CRYPTO.HASH_MD4	Message Digest 4 (MD4)
DBMS_CRYPTO.HASH_SH1	Secure Hashing Algorithm 1 (SHA-1)

Например, чтобы получить хеш-значение для переменной типа RAW, составим функцию :

```
CREATE OR REPLACE FUNCTION get_sha1_hash_val (p_in RAW)
RETURN RAW
IS
l_hash   RAW (4000);
BEGIN
l_hash := DBMS_CRYPTO.HASH (src => p_in, typ => DBMS_CRYPTO.HASH_SH1);
RETURN l_hash;
END;
```

Криптографическое хеширование

Код аутентификации сообщения (MAC) в Oracle 10g

Рассмотренный ранее метод хеширования весьма полезен, но имеет некоторые недостатки:

- проверить подлинность переданных данных с помощью хеш-функции может кто угодно. Это может оказаться недопустимым для некоторых систем повышенной безопасности, предполагающих, что аутентичность сообщения или данных проверяет только определенный получатель;
- узнав алгоритм хеширования, злоумышленник может вычислить правильное хеш-значение и подменить им реальное, скрыв тем самым изменение данных;
- по причине, указанной в предыдущем пункте, хранение хеш-значения вместе с данными не представляется безопасным. Каждый, кто имеет привилегию на изменение таблицы, сможет изменить и хеш-значение. Аналогично некто может сгенерировать хеш-значение и изменить данные «в пути». Поэтому хеш-значение не должно сопровождать данные, а обязательно должно передаваться отдельно, что усложняет систему.

Криптографическое хеширование

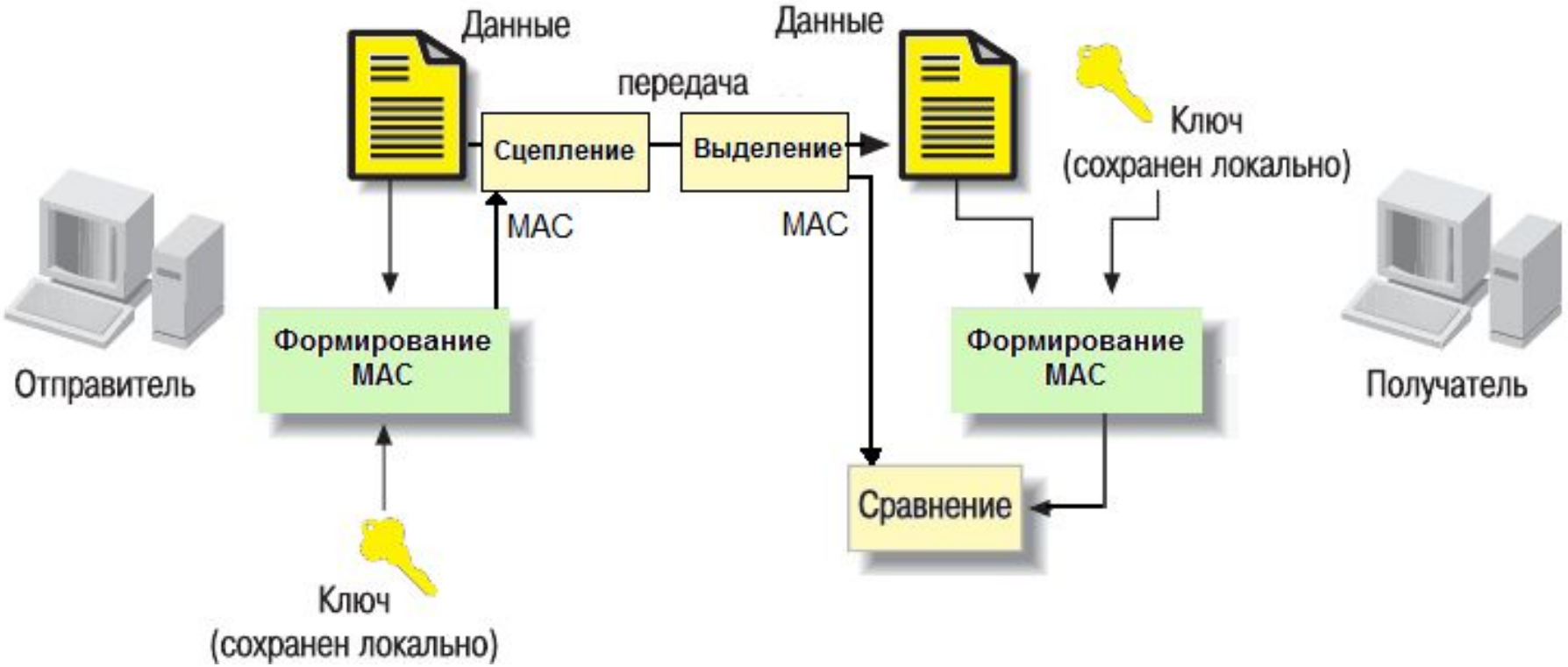


Рис. 6. – Использование кода аутентификации сообщения

Как и хеширование, MAC следует стандартным алгоритмам MD5 и SHA-1.
Пример функции, вычисляющей значение *MAC* для различных алгоритмов:

```
CREATE OR REPLACE FUNCTION get_mac_val (
   p_in_val       IN   VARCHAR2,
   p_key          IN   VARCHAR2,
   p_algorithm    IN   VARCHAR2 := 'SH1'
)
RETURN VARCHAR2
IS
l_mac_val     RAW (4000);
l_key         RAW (4000);
l_mac_algo    PLS_INTEGER;
l_in          RAW (4000);
l_ret         VARCHAR2 (4000);
BEGIN
   l_mac_algo :=
      CASE p_algorithm
         WHEN 'SH1'
            THEN DBMS_CRYPTO.hmac_sh1
         WHEN 'MD5'
            THEN DBMS_CRYPTO.hmac_md5
      END;
l_in := utl_i18n.string_to_raw (p_in_val, 'AL32UTF8');
l_key := utl_i18n.string_to_raw (p_key, 'AL32UTF8');
l_mac_val := DBMS_CRYPTO.mac (src => l_in, typ => l_mac_algo, key=>l_key);
l_ret := RAWTOHEX (l_mac_val);
RETURN l_ret;
END;
```