Криптография — наука о методах обеспечения конфиденциальности, целостности данных, аутентификации, а также невозможности отказа от авторства.

ОСНОВНЫЕ ЦЕЛИ КРИПТОГРАФИИ

Обеспечение:

1. Конфиденциальности.

2. Аутентификации.

3. Целостности.

4. Невозможности отказа.

ОСНОВНЫЕ МЕТОДЫ ШИФРОВАНИЯ ИНФОРМАЦИИ

1. Симметричное шифрование
   - Поточные шифры
   - Блочные шифры
   - Хэш-функции
2. Асимметричное шифрование
   - Протоколы обмена ключами
   - Асимметричные (несимметричные) шифры
   - Цифровая подпись

1. Симметричные криптографические системы

- Основные классы симметричных криптографических систем.
- Стойкость криптографических систем и алгоритмов.
- Основные методы криптоанализа шифров.
- Простейшие шифры и их основные свойства.
- Поточные шифры: основные требования, режимы функционирования.
- Блочные шифры: основные требования к стойкости и режимы функционирования.
- Основные структуры блочных шифров: SP-сети и сети Фейстеля.
- Российский стандарт блочного шифрования «Кузнечик»: основные параметры, структура раунда шифрования и принцип действия, процедура разворачивания исходного ключа в раундовые (рабочие) подключи, режимы использования.
- Стойкость, обеспечиваемая современными шифрами. Причины ненадёжности криптографических систем.

1.1. Основные классы симметричных криптографических систем.

1. Подстановки.
2. Перестановки.
3. Поточные шифры.
4. Блочные шифры.
5. Хэш-функции.

Симметричные криптосистемы – использование одинакового секретного ключа как для шифрования, так и для расшифрования информации.

1.2. Стойкость криптографических систем и алгоритмов.

Два вида шифров:

1. Шифры, созданные в эпоху донаучной криптографии (шифр Цезаря, шифры подстановки и перестановки и др.).
2. Шифры, созданные в эпоху научной криптографии (с середины 40-х годов 20 века, после трудов Клода Шеннона).

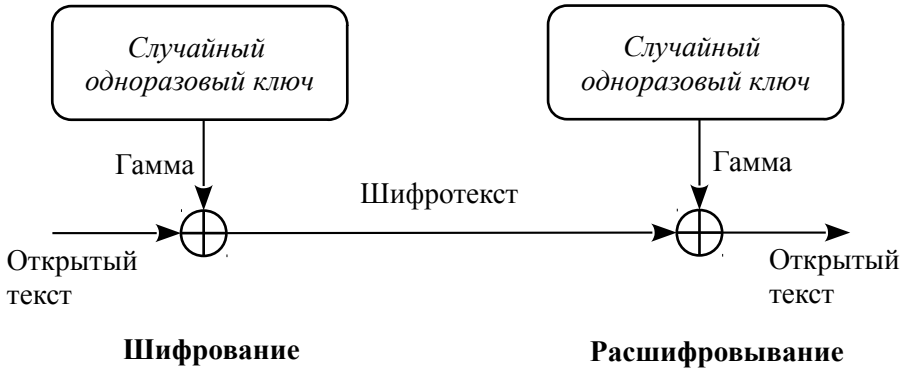
Два класса стойкости:

1. Теоретически недешифруемые шифры.
2. Практически (вычислительно) стойкие шифры.

1. Теоретически недешифруемые шифры.

Шифр Вернама (одноразовый гамма-блокнот, 1917 год) – длина ключа совпадает с длиной сообщения, ключ одноразовый и заранее формируется случайным образом (и распределяется между абонентами).

Даже при условии наличия у злоумышленника *бесконечных вычислительных ресурсов* шифр не может быть взломан, так как все варианты ключей равновероятны и равноценны. Нет возможности определить, какой именно ключ верный.



Доказательство стойкости шифра Вернама (1945 год) – важнейший вклад **Клода Шеннона** в становление и развитие научной криптографии и важнейшее приложение предложенной им **Теории информации**.

Основной недостаток – проблема распределения и хранения большого объёма секретных ключей.

2. Практически (вычислительно) стойкие шифры – все современные шифры.

Основа появления – устранить основной недостаток одноразового гамма-блокнота – формирование, распределение и хранение большого объёма ключевой информации.

Поточные шифры – имитация одноразового гамма-блокнота, при этом, ключевая гамма формируется из *короткого многоразового* исходного ключа.

Теоретическая основа – теория сложности вычислений (существование NP-полных задач).

Стойкость сводится к количеству комбинаций, которые необходимо перебрать для определения правильного ключа (или, исходного открытого сообщения) – в случае, если шифр не имеет уязвимостей.

длина ключа	количество комбинаций	время на полный перебор, дней*
56 бит	$7{,}205759404\times10^{16}$	0,00001668
64 бит	$1{,}844674407\times10^{19}$	0,00427008
80 бит	$1{,}20892582\times10^{24}$	279,8
128 бит	$3{,}402823669\times10^{38}$	$7{,}876906642\times10^{16}$
256 бит	$1{,}157920892\times10^{77}$	$2{,}680372436\times10^{55}$

(* при скорости перебора 1 000 000 000 × 50×10^{6} или 5×10^{16} ключей в секунду)

1.3. Основные методы криптоанализа шифров.

Криптоанализ – наука о методах определения зашифрованной информации без знания ключа. Криптоанализ включает также методы выявления уязвимости криптографических алгоритмов или протоколов.

Криптографическая атака – попытка раскрытия конкретного шифра с применением методов криптоанализа.

Взлом или **вскрытие** шифра – криптографическая атака, в ходе которой удалось раскрыть шифр.

Основные методы криптоанализа:

- Атака на основе шифротекста
- Атака на основе открытых текстов и соответствующих шифротекстов
- Атака на основе подобранного открытого текста
- Атака на основе адаптивно подобранного открытого текста

Для современных шифров используется следующее предположение – структура и описание шифра известны, *неизвестным* является только *ключ шифрования*.

Различные виды криптографических атак

Типовые атаки (подходят для любых симметричных криптоалгоритмов):

1. Полный перебор ключей (метод грубой силы).
2. Восстановление кодовой книги (для блочных шифров).
3. Атака на основе коллизий (атака «дней рождения», для блочных шифров и хэш-функций).

Основные специфичные методы (разрабатываются под конкретные криптоалгоритмы).

1. Частотный криптоанализ.
2. Линейный криптоанализ.
3. Дифференциальный криптоанализ и его производные (атаки на основе полных дифференциалов, усечённых дифференциалов, невозможных дифференциалы и др.).
4. Интегральный криптоанализ (обобщённый вид дифференциального криптоанализа).
5. Алгебраический криптоанализ.
6. Интерполяционная атака.
7. Атака по сторонним каналам утечки информации (анализ потребляемой мощности, времени выполнения операций и др.)

Задача криптоаналитика – поиск атаки на шифр, сложность которой меньше, чем типовых атак.

1.4. Простейшие шифры и их основные свойства

Простейшие шифры не являются стойкими.

1. Шифры замены (подстановки) – в соответствии с заданным секретным правилом (таблицей замены или подстановки) каждый символ в тексте заменяется на другой.
2. Шифры перестановки – в соответствии с заданным секретным правилом происходит перестановка символов в тексте.

Примеры известных шифров:

- Шифр Цезаря
- Шифр пар
- Шифр четырех квадратов
- Матричный шифр
- Шифр ADFGX
- Шифр Виженера

Примеры:

Исходный текст:

CrypTool 1 (CT1) is a comprehensive and free educational program
about cryptography and cryptanalysis
offering extensive online help and many visualizations.

Результат шифрования:

Шифр Цезаря (ROT-3)

Правило замены:

ABCDEFGHIJKLMNOPQRSTUVWXYZ

CDEFGHIJKLMNOPQRSTUVWXYZAB

EtarVqqn 1 (EV1) ku c eqortgjgpukxg cpf htgg gfwecvkqpcn rtqitco
cdqwv etarvqitcrja cpf etarvcpcnauku
qhhgtkpi gzvgpukxg qpnkpg jgnr cpf ocpa xkuwcnkbcvkqpu.

Шифр Вижинера

Нумерация букв в алфавите:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z

Ключ: **ictis**

Шифрование – побуквенное сложение по модулю N (N = 26 для заданного алфавита).

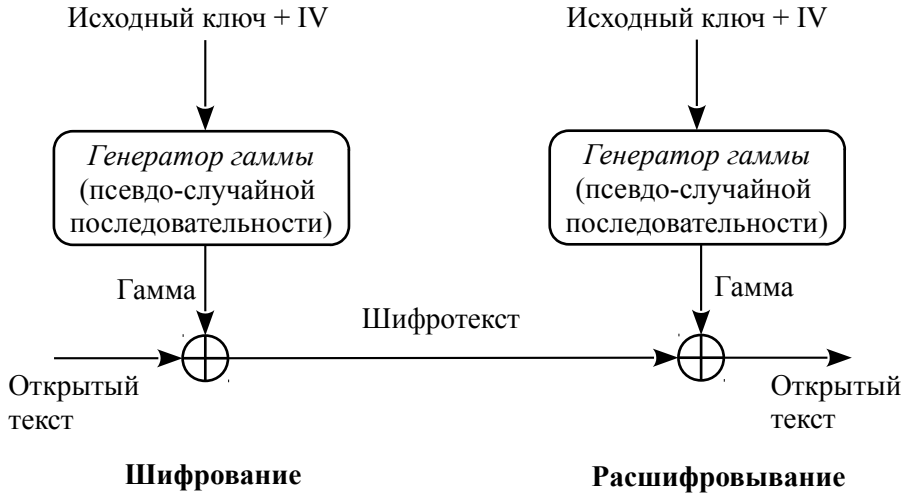
```
offering      14 5 5  4  17 8 13 6
ictisict      8  2 19 8  18 8 2  19
              22 7 24 12 35 8 15 25
              22 7 24 12 9  8 15 25

whymjipz
```

1.5. Поточные шифры: основные требования, режимы функционирования

Поточный шифр – это имитация одноразового гамма-блокнота, стойкость ограничена длиной ключа.

Основа поточного шифра – генератор псевдослучайной последовательности (ГПСП). Качество формируемой псевдослучайной последовательности (гаммы) определяет стойкость шифра.



Идеальный случай – гамма неотличима от случайной последовательности. В реальности, для обеспечения возможности расшифрования последовательность должна быть псевдослучайной, т. е. воспроизводимой (формируется конечным автоматом).

Тестирование на случайность – тесты NIST STS, детальный анализ структуры шифра.

Режимы функционирования:

1. Синхронные.
2. Самосинхронизирующиеся.

Основные способы построения ГПСП:

1. Конгруэнтные генераторы (нестойкие) – часто применяются как ГПСП в программных библиотеках.
2. На основе регистров сдвига с линейной обратной связью (в большинстве случаев, взломаны. A5/2).
3. На основе блочных шифров.
4. Другие (RC4).

Достоинства:

1. Отсутствие эффекта распространения ошибок.
2. Высокая скорость и простота (зачастую, в ущерб стойкости).
3. Для шифрования и расшифрования используются идентичные блоки ГПСП.

Недостатки:

1. Отсутствие возможности генерации ПСП с произвольного места.

2. Если не используется вектор инициализации (IV), каждый раз гамма будет одинаковой, что открывает возможность простой атаки (известные уязвимости в продуктах Microsoft Office и др.).

3. В настоящее время практически вытеснены блочными шифрами.

Рекомендованные поточные шифры:

- Trivium
- Salsa20/8
- HC-128

1.6. Блочные шифры: основные требования к стойкости и режимы функционирования.

Режимы функционирования:

1. Электронная кодовая книга (Electronic code book, ECB).
2. Режим сцепления блоков шифротекста (Cipher block chaining, CBC)
3. Режим обратной связи по шифротексту (Cipher feed back, CFB)
4. Режим обратной связи по выходу (Output feed back, OFB)
5. Режим счётчика (Counter mode, CTR)

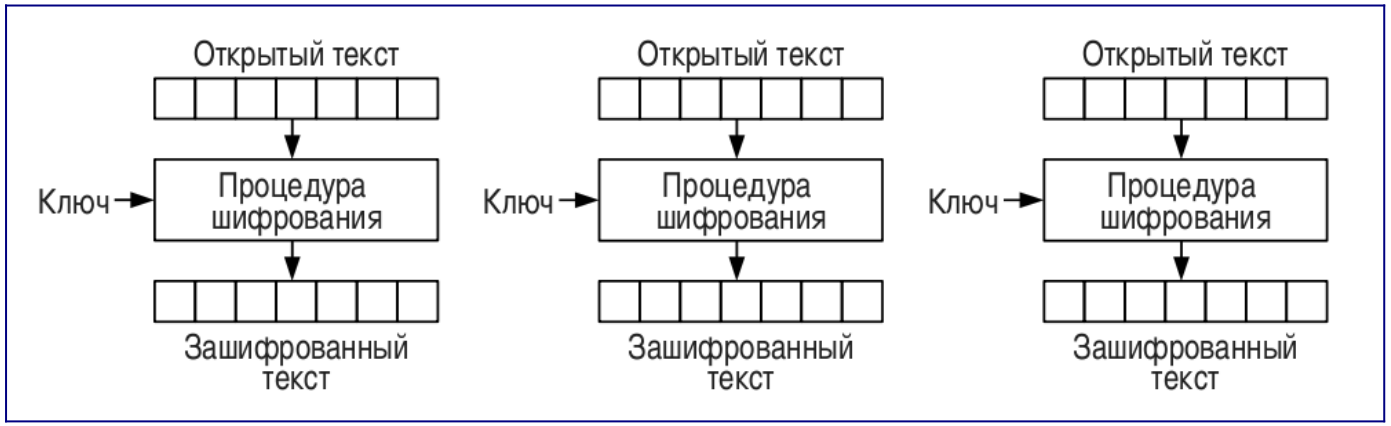
1. Электронная кодовая книга – каждый блок открытого текста заменяется блоком шифротекста. В ГОСТ 28147—89 называется режимом простой замены.

Шифрование может быть описано следующим образом:

$$C_i = E_k\left(P_i\right)$$

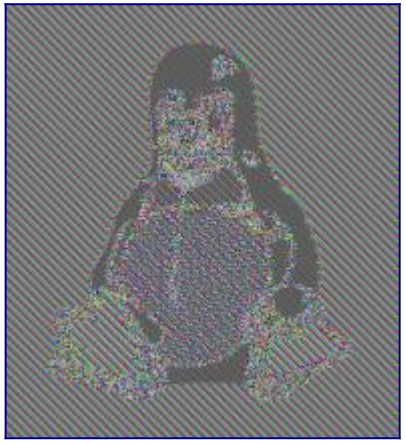
где i – номера блоков, Ci и Pi — блоки зашифрованного и открытого текстов соответственно, Ek – функция блочного шифрования. Расшифрование:

$$P_i = D_k\left(C_i\right)$$

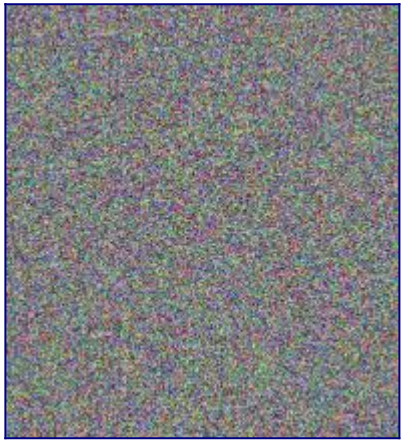


Пример (https://ru.wikipedia.org/wiki):

Открытый текст в виде изображения

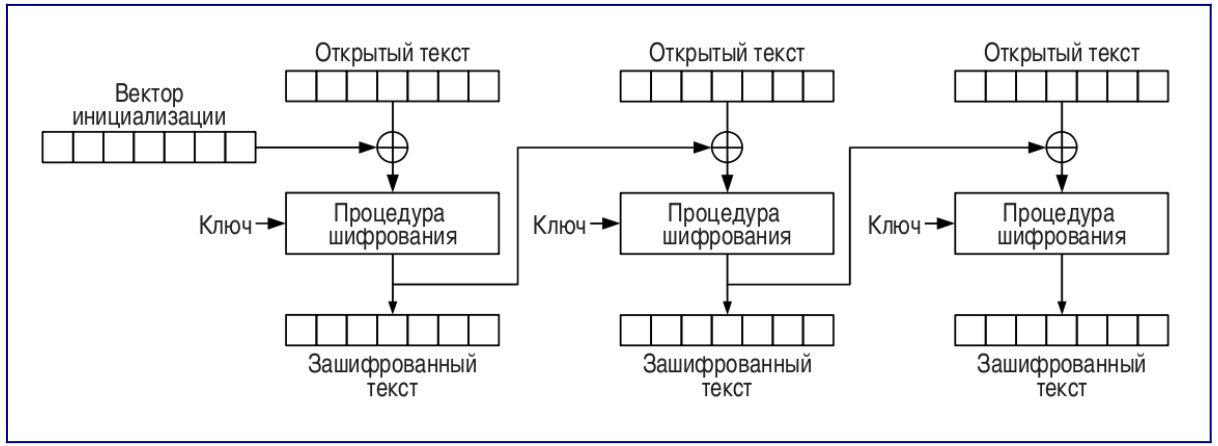


Криптограмма, полученная шифрованием в режиме ECB.

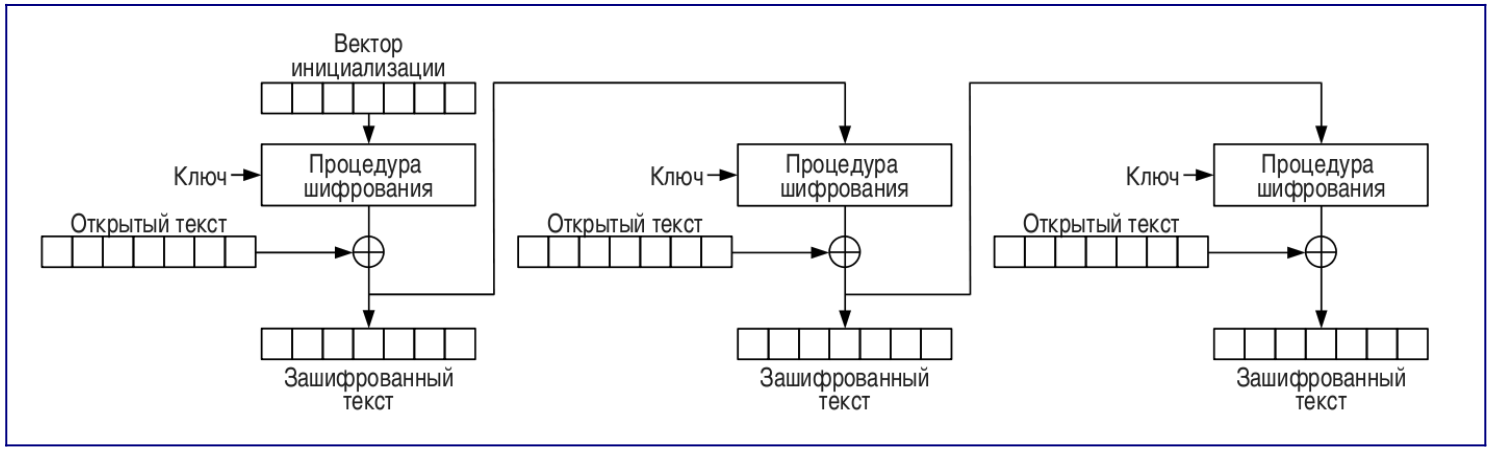


Криптограмма, полученная шифрованием в режиме, отличном от ECB.

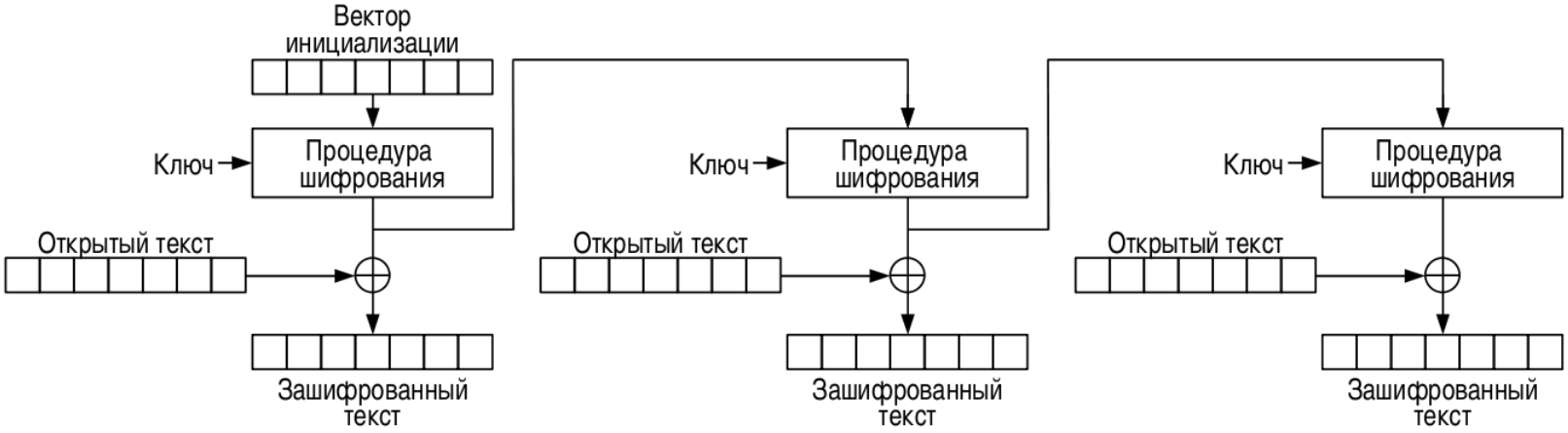
2. Режим сцепления блоков шифротекста (Cipher block chaining, CBC)



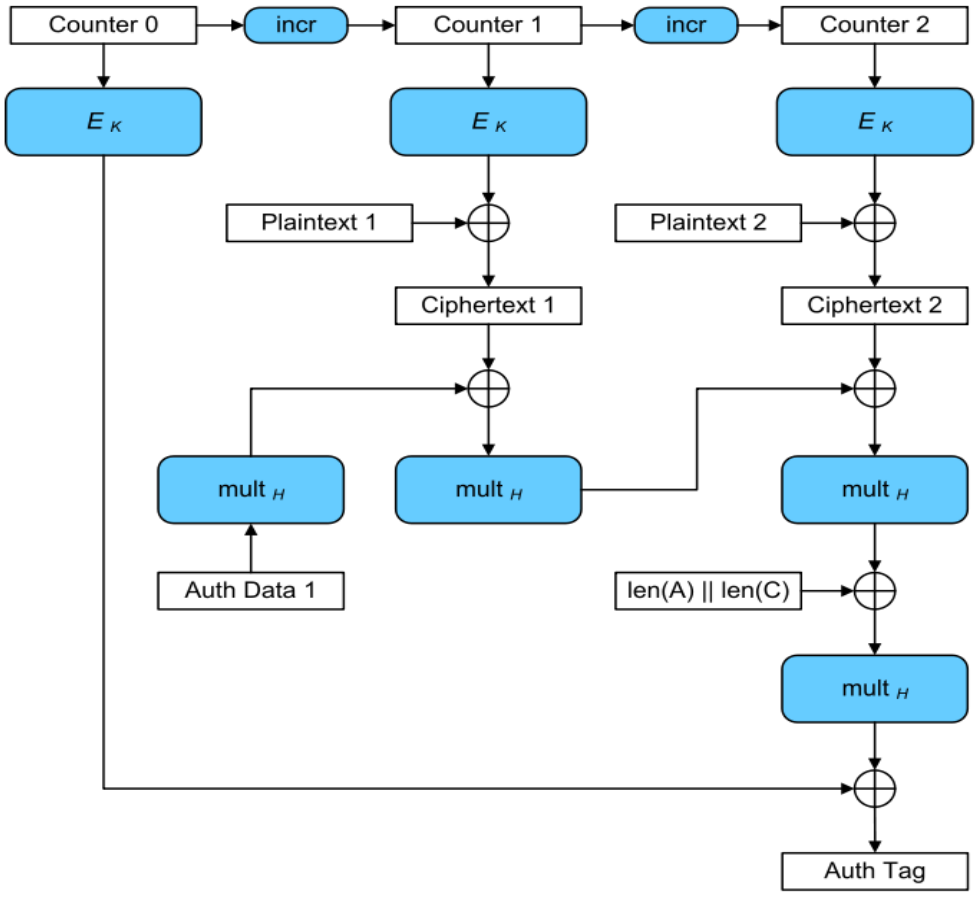
3. Режим обратной связи по шифротексту (Cipher feed back, CFB)



4. Режим обратной связи по выходу (Output feed back, OFB)



5. Режим счётчика (Counter mode, CTR)



Основные требования к стойкости

Длина ключа:

- 80 бит (для малоресурсной криптографии)
- 128 бит
- 256 бит

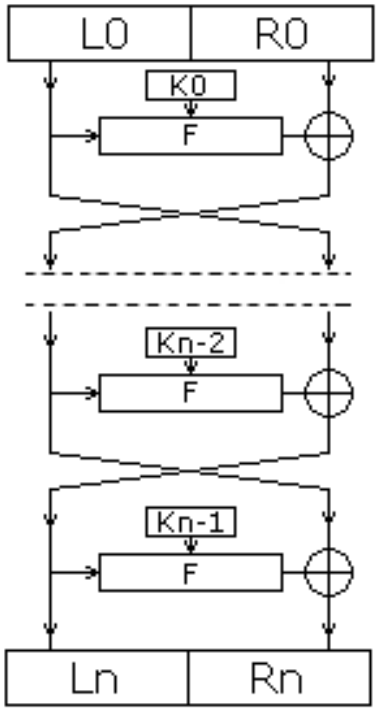
Длина блока (защита от создания словаря и обеспечение периода ПСП):

- 64 бит (шифр Магма, недостаточно против атаки на основе коллизий)
- 128 бит
- 256 бит

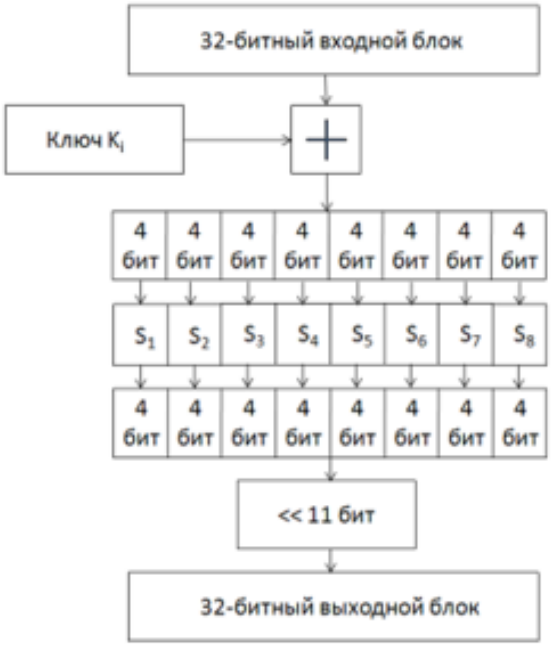
Устойчивость к различным видам криптоанализа. Если сложность любой из криптографических атак меньше полного перебора, то шифр считается нестойким.

1.7. Основные структуры блочных шифров: SP-сети и сети Фейстеля.

Сеть Фейстеля (англ. Feistel network, Feistel cipher) — метод построения блочных шифров из однотипных блоков, одинаковых как при шифровании, так и расшифровании (изменяется только порядок следования раундовых ключей).



Пример функции преобразования для шифра Магма:



Достоинства:

1. Функция преобразования может быть невзаимооднозначной.

2. Одинаковые функции преобразования как для шифрования, так и для расшифрования (требует меньше ресурсов).

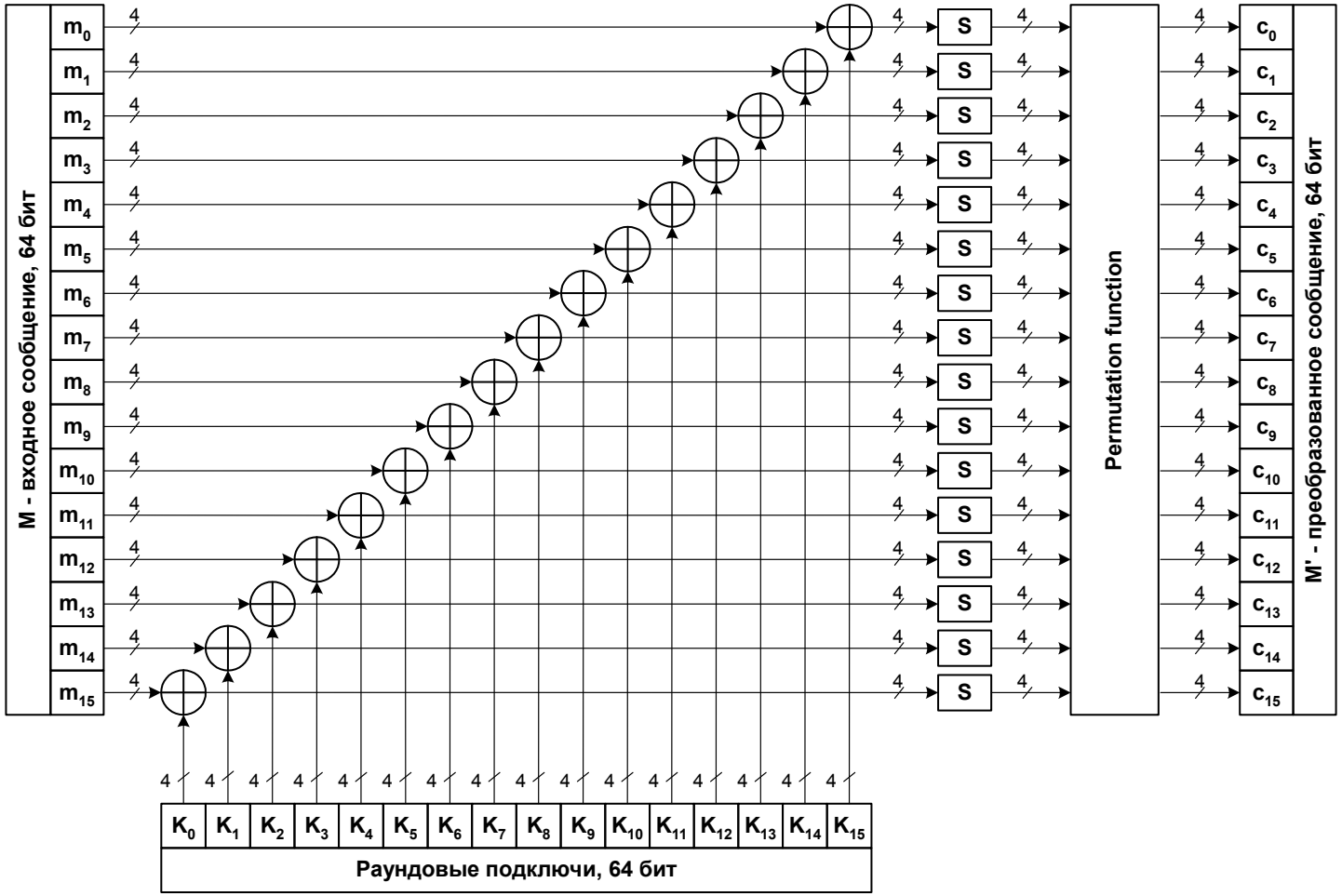
3. Наличие специфических атак (например, Yo-Yo-games)

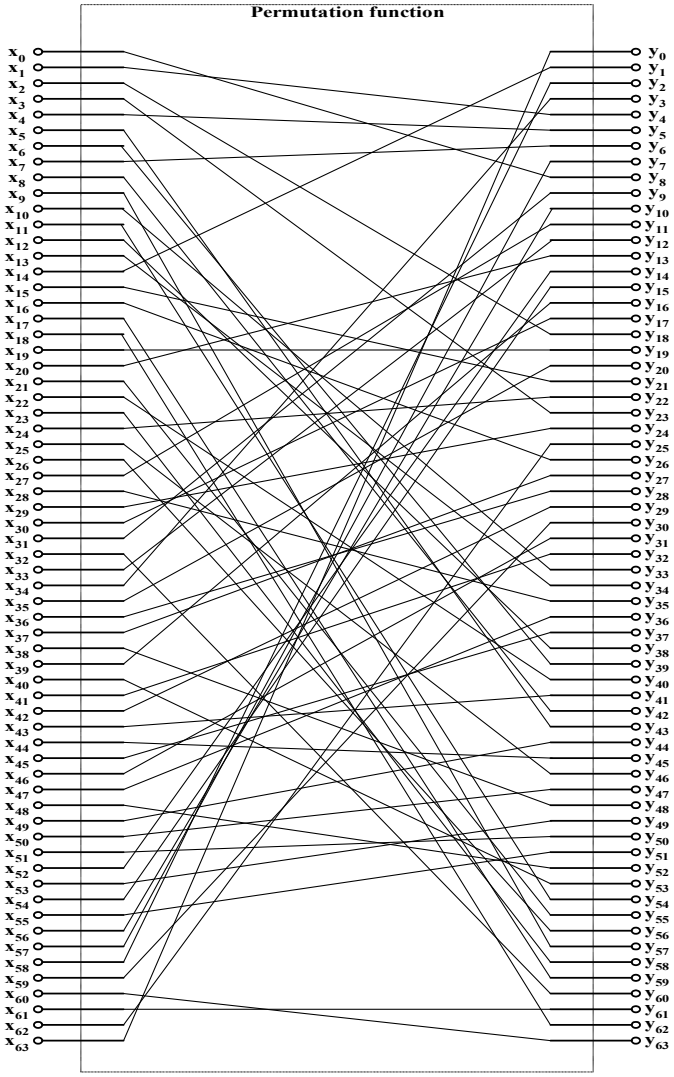
Недостаток:

1. Требует больше раундов шифрования (вследствие меньшей скорости накопления сложности преобразований).

Примеры блочных шифров: DES, RC2, RC5, RC6, Blowfish, FEAL, CAST-128, TEA, XTEA, XXTEA и др.

SP-сеть (Substitution-Permutation network, подстановочно-перестановочная сеть) – в простейшем варианте представляет собой «сэндвич» из слоёв двух типов, используемых многократно по очереди. Большинство современных блочных шифров построены на основе SP-сети.

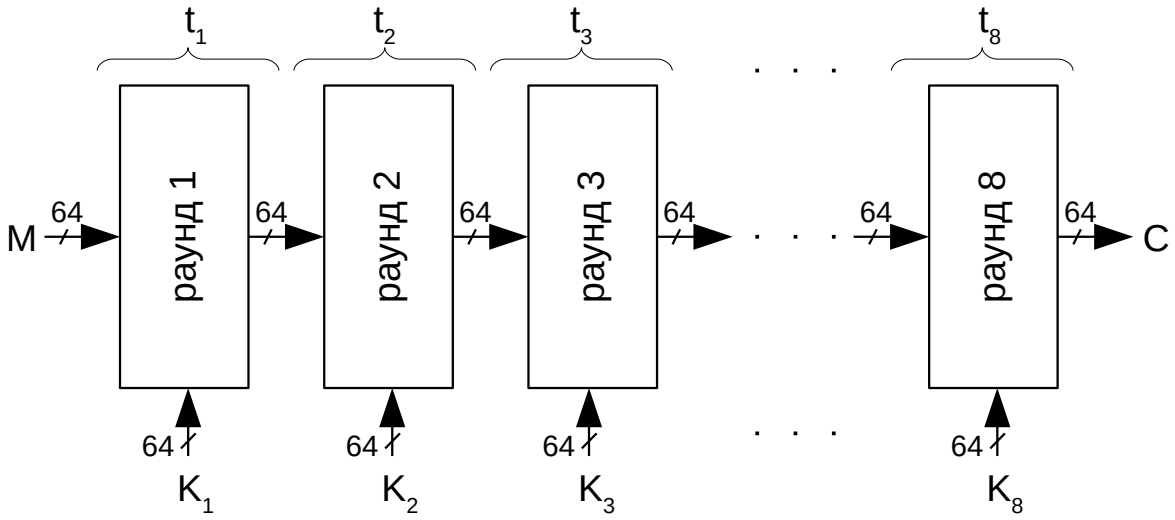



Permutation function
x0 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16 x17 x18 x19 x20 x21 x22 x23 x24 x25 x26 x27 x28 x29 x30 x31 x32 x33 x34 x35 x36 x37 x38 x39 x40 x41 x42 x43 x44 x45 x46 x47 x48 x49 x50 x51 x52 x53 x54 x55 x56 x57 x58 x59 x60 x61 x62 x63
y0 y1 y2 y3 y4 y5 y6 y7 y8 y9 y10 y11 y12 y13 y14 y15 y16 y17 y18 y19 y20 y21 y22 y23 y24 y25 y26 y27 y28 y29 y30 y31 y32 y33 y34 y35 y36 y37 y38 y39 y40 y41 y42 y43 y44 y45 y46 y47 y48 y49 y50 y51 y52 y53 y54 y55 y56 y57 y58 y59 y60 y61 y62 y63

Пример подстановки (4 бит вход и выход):

sbox(X): {14, 5, 11, 7, 4, 13, 3, 10, 9, 2, 1, 15, 8, 6, 0, 12}

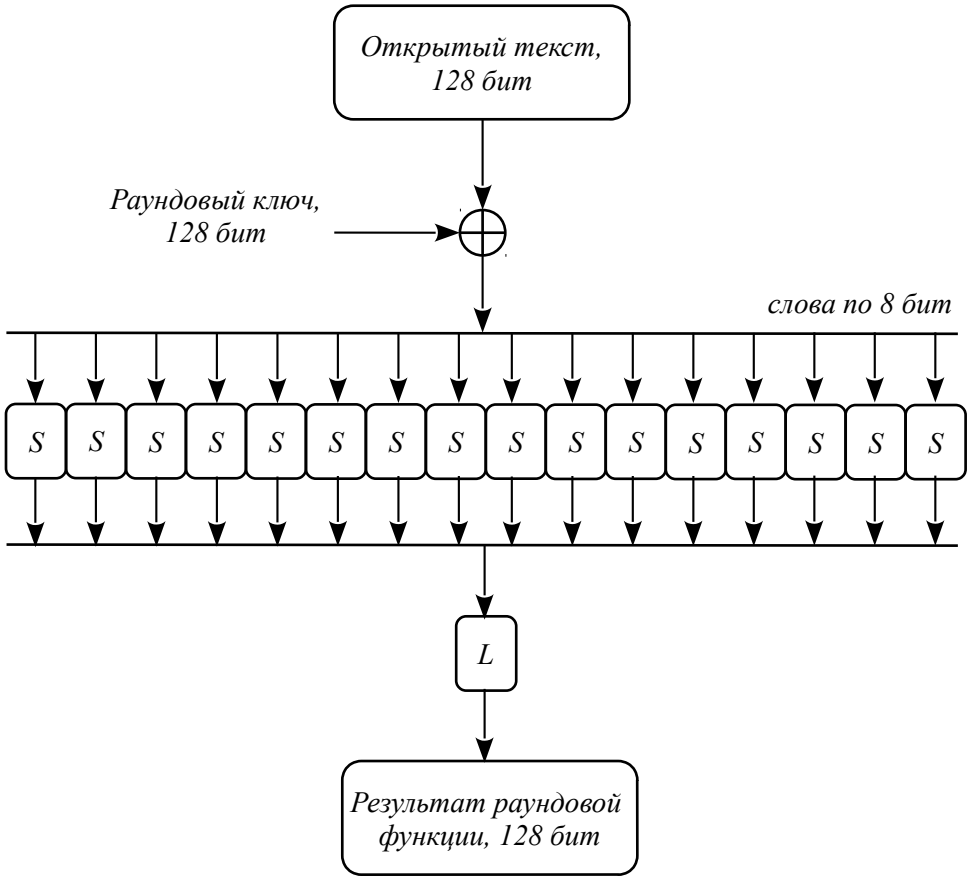
Раундовая структура шифра:



Примеры блочных шифров:

- AES – 2000 год (Rijndael), SP-сеть, длина ключа {128, 192, 256} бит, размер блока {128, 192, 256} бит, количество раундов – {12, 14, 16}.
- Кузнечик – 2015 год (tk26.ru), SP-сеть, длина ключа 256 бит, размер блока 128 бит, количество раундов – 10.

1.8. Российский стандарт блочного шифрования «Кузнечик» (ГОСТ Р 34.12-2015): основные параметры, структура раунда шифрования и принцип действия, процедура разворачивания исходного ключа в раундовые (рабочие) подключи, режимы использования.



Исходный код «Кузнечик» на примере реализации от Markku-Juhani O. Saarinen: (https://github.com/mjosaarinen/kuznechik)

```
// encrypt a block - 8 bit way

void kuz_encrypt_block(kuz_key_t *key, void *blk)
{
   int i, j;
   w128_t x;

   x.q[0] = ((uint64_t *) blk)[0];
   x.q[1] = ((uint64_t *) blk)[1];

   for (i = 0; i < 9; i++) {

      x.q[0] ^= key->k[i].q[0];
      x.q[1] ^= key->k[i].q[1];

      for (j = 0; j < 16; j++)
         x.b[j] = kuz_pi[x.b[j]];

      kuz_l(&x);
   }
   ((uint64_t *) blk)[0] = x.q[0] ^ key->k[9].q[0];
   ((uint64_t *) blk)[1] = x.q[1] ^ key->k[9].q[1];

}
```

```
// The S-Box from section 5.1.1
const uint8_t kuz_pi[0x100] = {
   0xFC, 0xEE, 0xDD, 0x11, 0xCF, 0x6E, 0x31, 0x16,   // 00..07
   0xFB, 0xC4, 0xFA, 0xDA, 0x23, 0xC5, 0x04, 0x4D,   // 08..0F
   0xE9, 0x77, 0xF0, 0xDB, 0x93, 0x2E, 0x99, 0xBA,   // 10..17
   0x17, 0x36, 0xF1, 0xBB, 0x14, 0xCD, 0x5F, 0xC1,   // 18..1F
   0xF9, 0x18, 0x65, 0x5A, 0xE2, 0x5C, 0xEF, 0x21,   // 20..27
   0x81, 0x1C, 0x3C, 0x42, 0x8B, 0x01, 0x8E, 0x4F,   // 28..2F
   0x05, 0x84, 0x02, 0xAE, 0xE3, 0x6A, 0x8F, 0xA0,   // 30..37
   0x06, 0x0B, 0xED, 0x98, 0x7F, 0xD4, 0xD3, 0x1F,   // 38..3F
   0xEB, 0x34, 0x2C, 0x51, 0xEA, 0xC8, 0x48, 0xAB,   // 40..47
   0xF2, 0x2A, 0x68, 0xA2, 0xFD, 0x3A, 0xCE, 0xCC,   // 48..4F
   0xB5, 0x70, 0x0E, 0x56, 0x08, 0x0C, 0x76, 0x12,   // 50..57
   0xBF, 0x72, 0x13, 0x47, 0x9C, 0xB7, 0x5D, 0x87,   // 58..5F
   0x15, 0xA1, 0x96, 0x29, 0x10, 0x7B, 0x9A, 0xC7,   // 60..67
   0xF3, 0x91, 0x78, 0x6F, 0x9D, 0x9E, 0xB2, 0xB1,   // 68..6F
   0x32, 0x75, 0x19, 0x3D, 0xFF, 0x35, 0x8A, 0x7E,   // 70..77
   0x6D, 0x54, 0xC6, 0x80, 0xC3, 0xBD, 0x0D, 0x57,   // 78..7F
   0xDF, 0xF5, 0x24, 0xA9, 0x3E, 0xA8, 0x43, 0xC9,   // 80..87
   0xD7, 0x79, 0xD6, 0xF6, 0x7C, 0x22, 0xB9, 0x03,   // 88..8F
   0xE0, 0x0F, 0xEC, 0xDE, 0x7A, 0x94, 0xB0, 0xBC,   // 90..97
   0xDC, 0xE8, 0x28, 0x50, 0x4E, 0x33, 0x0A, 0x4A,   // 98..9F
   0xA7, 0x97, 0x60, 0x73, 0x1E, 0x00, 0x62, 0x44,   // A0..A7
   0x1A, 0xB8, 0x38, 0x82, 0x64, 0x9F, 0x26, 0x41,   // A8..AF
   0xAD, 0x45, 0x46, 0x92, 0x27, 0x5E, 0x55, 0x2F,   // B0..B7
   0x8C, 0xA3, 0xA5, 0x7D, 0x69, 0xD5, 0x95, 0x3B,   // B8..BF
   0x07, 0x58, 0xB3, 0x40, 0x86, 0xAC, 0x1D, 0xF7,   // C0..C7
   0x30, 0x37, 0x6B, 0xE4, 0x88, 0xD9, 0xE7, 0x89,   // C8..CF
   0xE1, 0x1B, 0x83, 0x49, 0x4C, 0x3F, 0xF8, 0xFE,   // D0..D7
   0x8D, 0x53, 0xAA, 0x90, 0xCA, 0xD8, 0x85, 0x61,   // D8..DF
   0x20, 0x71, 0x67, 0xA4, 0x2D, 0x2B, 0x09, 0x5B,   // E0..E7
   0xCB, 0x9B, 0x25, 0xD0, 0xBE, 0xE5, 0x6C, 0x52,   // E8..EF
   0x59, 0xA6, 0x74, 0xD2, 0xE6, 0xF4, 0xB4, 0xC0,   // F0..F7
   0xD1, 0x66, 0xAF, 0xC2, 0x39, 0x4B, 0x63, 0xB6,   // F8..FF
};
```

```
// Linear vector from sect 5.1.2

static const uint8_t kuz_lvec[16] = {
   0x94, 0x20, 0x85, 0x10, 0xC2, 0xC0, 0x01, 0xFB,
   0x01, 0xC0, 0xC2, 0x10, 0x85, 0x20, 0x94, 0x01
};

// poly multiplication mod p(x) = x^8 + x^7 + x^6 + x + 1
// totally not constant time

static uint8_t kuz_mul_gf256(uint8_t x, uint8_t y)
{
   uint8_t z;

   z = 0;
   while (y) {
      if (y & 1)
         z ^= x;
      x = (x << 1) ^ (x & 0x80 ? 0xC3 : 0x00);
      y >>= 1;
   }

   return z;
}
```

```
// linear operation l

static void kuz_l(w128_t *w)
{
   int i, j;
   uint8_t x;

   // 16 rounds
   for (j = 0; j < 16; j++) {

      // An LFSR with 16 elements from GF(2^8)
      x = w->b[15];  // since lvec[15] = 1

      for (i = 14; i >= 0; i--) {
         w->b[i + 1] = w->b[i];
         x ^= kuz_mul_gf256(w->b[i], kuz_lvec[i]);
      }
      w->b[0] = x;
   }
}
```

```
// key setup

void kuz_set_encrypt_key(kuz_key_t *kuz, const uint8_t key[32])
{
   int i, j;
   w128_t c, x, y, z;

   for (i = 0; i < 16; i++) {
      // this will be have to changed for little-endian systems
      x.b[i] = key[i];
      y.b[i] = key[i + 16];
   }

   kuz->k[0].q[0] = x.q[0];
   kuz->k[0].q[1] = x.q[1];
   kuz->k[1].q[0] = y.q[0];
   kuz->k[1].q[1] = y.q[1];

   for (i = 1; i <= 32; i++) {

      // C Value
      c.q[0] = 0;
      c.q[1] = 0;
      c.b[15] = i;    // load round in lsb
      kuz_l(&c);

      z.q[0] = x.q[0] ^ c.q[0];
      z.q[1] = x.q[1] ^ c.q[1];
      for (j = 0; j < 16; j++)
         z.b[j] = kuz_pi[z.b[j]];
      kuz_l(&z);
```

```
      z.q[0] ^= y.q[0];
      z.q[1] ^= y.q[1];

      y.q[0] = x.q[0];
      y.q[1] = x.q[1];

      x.q[0] = z.q[0];
      x.q[1] = z.q[1];

      if ((i & 7) == 0) {
         kuz->k[(i >> 2)].q[0] = x.q[0];
         kuz->k[(i >> 2)].q[1] = x.q[1];
         kuz->k[(i >> 2) + 1].q[0] = y.q[0];
         kuz->k[(i >> 2) + 1].q[1] = y.q[1];
      }
   }
}
```

```
// decrypt a block - 8 bit way

void kuz_decrypt_block(kuz_key_t *key, void *blk)
{
   int i, j;
   w128_t x;

   x.q[0] = ((uint64_t *) blk)[0] ^ key->k[9].q[0];
   x.q[1] = ((uint64_t *) blk)[1] ^ key->k[9].q[1];

   for (i = 8; i >= 0; i--) {

      kuz_l_inv(&x);
      for (j = 0; j < 16; j++)
         x.b[j] = kuz_pi_inv[x.b[j]];

      x.q[0] ^= key->k[i].q[0];
      x.q[1] ^= key->k[i].q[1];
   }
   ((uint64_t *) blk)[0] = x.q[0];
   ((uint64_t *) blk)[1] = x.q[1];
}
```

Пример шифрования информации:

K_1 = 88 99 AA BB CC DD EE FF 00 11 22 33 44 55 66 77

K_2 = FE DC BA 98 76 54 32 10 01 23 45 67 89 AB CD EF

K_3 = DB 31 48 53 15 69 43 43 22 8D 6A EF 8C C7 8C 44

K_4 = 3D 45 53 D8 E9 CF EC 68 15 EB AD C4 0A 9F FD 04

K_5 = 57 64 64 68 C4 4A 5E 28 D3 E5 92 46 F4 29 F1 AC

K_6 = BD 07 94 35 16 5C 64 32 B5 32 E8 28 34 DA 58 1B

K_7 = 51 E6 40 75 7E 87 45 DE 70 57 27 26 5A 00 98 B1

K_8 = 5A 79 25 01 7B 9F DD 3E D7 2A 91 A2 22 86 F9 84

K_9 = BB 44 E2 53 78 C7 31 23 A5 F3 2F 73 CD B6 E5 17

K_10 = 72 E9 DD 74 16 BC F4 5B 75 5D BA A8 8E 4A 40 43

PT = 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

ENCRYPT

PLAIN TEXT 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

ROUND KEY (0) 88 99 AA BB CC DD EE FF 00 11 22 33 44 55 66 77

result X 88 99 AA BB CC DD EE FF 00 11 22 33 44 55 66 77

result PI D7 E8 38 7D 88 D8 6C B6 FC 77 65 AE EA 0C 9A 7E

L (15) 57 E3 90 2E CA 85 E9 FF 92 8F B3 E7 08 6F 50 BC

result L 57 E3 90 2E CA 85 E9 FF 92 8F B3 E7 08 6F 50 BC

ROUND KEY (1) FE DC BA 98 76 54 32 10 01 23 45 67 89 AB CD EF

result X A9 3F 2A B6 BC D1 DB EF 93 AC F6 80 81 C4 9D 53
result PI B8 1F 3C 55 69 1B 90 52 DE 64 B4 DF F5 86 33 56
L (15) 36 4A 31 51 A1 85 50 AE 81 7B 79 1C E6 D5 B5 C4
result L 36 4A 31 51 A1 85 50 AE 81 7B 79 1C E6 D5 B5 C4
ROUND KEY (2) DB 31 48 53 15 69 43 43 22 8D 6A EF 8C C7 8C 44
result X ED 7B 79 02 B4 EC 13 ED A3 F6 13 F3 6A 12 39 80
result PI E5 80 54 DD 27 BE DB E5 73 B4 DB D2 78 F0 0B DF
L (15) 13 8D A9 BA D0 A1 82 EE 9C D4 04 DA E2 42 CE 5E
result L 13 8D A9 BA D0 A1 82 EE 9C D4 04 DA E2 42 CE 5E
ROUND KEY (3) 3D 45 53 D8 E9 CF EC 68 15 EB AD C4 0A 9F FD 04
result X 2E C8 FA 62 39 6E 6E 86 89 3F A9 1E E8 DD 33 5A
result PI 8E 30 AF 96 0B B2 B2 43 79 1F B8 5F CB D8 AE 13
L (15) 5B E4 73 84 7E 54 AC 3A 91 72 5D 03 BC 09 86 D8
result L 5B E4 73 84 7E 54 AC 3A 91 72 5D 03 BC 09 86 D8
ROUND KEY (4) 57 64 64 68 C4 4A 5E 28 D3 E5 92 46 F4 29 F1 AC
result X 0C 80 17 EC BA 1E F2 12 42 97 CF 45 48 20 77 74
result PI 23 DF BA BE A5 5F 74 F0 2C BC 89 C8 F2 F9 7E FF
L (15) 8F 79 60 C6 C0 60 80 EA 47 FC F3 5B 3F 60 11 92
result L 8F 79 60 C6 C0 60 80 EA 47 FC F3 5B 3F 60 11 92
ROUND KEY (5) BD 07 94 35 16 5C 64 32 B5 32 E8 28 34 DA 58 1B
result X 32 7E F4 F3 D6 3C E4 D8 F2 CE 1B 73 0B BA 49 89
result PI 02 0D E6 D2 F8 7F 2D 8D 74 E7 BB 3D DA A5 2A 79
L (15) 1F FA 42 E2 92 14 04 FF A1 75 90 81 B7 D8 A5 C1

result L 1F FA 42 E2 92 14 04 FF A1 75 90 81 B7 D8 A5 C1

ROUND KEY (6) 51 E6 40 75 7E 87 45 DE 70 57 27 26 5A 00 98 B1

result X 4E 1C 02 97 EC 93 41 21 D1 22 B7 A7 ED D8 3D 70

result PI CE 14 DD BC BE DE 34 18 1B 65 2F 44 E5 8D D4 32

L (15) 1D B0 35 89 7A 50 C4 81 A2 A1 72 98 C4 71 74 98

result L 1D B0 35 89 7A 50 C4 81 A2 A1 72 98 C4 71 74 98

ROUND KEY (7) 5A 79 25 01 7B 9F DD 3E D7 2A 91 A2 22 86 F9 84

result X 47 C9 10 88 01 CF 19 BF 75 8B E3 3A E6 F7 8D 1C

result PI AB 37 E9 D7 EE 89 36 3B 35 F6 A4 ED 09 C0 22 14

L (15) 1A 15 85 36 92 D4 23 A1 45 02 A5 4B 6A 4B 0B 4B

result L 1A 15 85 36 92 D4 23 A1 45 02 A5 4B 6A 4B 0B 4B

ROUND KEY (8) BB 44 E2 53 78 C7 31 23 A5 F3 2F 73 CD B6 E5 17

result X A1 51 67 65 EA 13 12 82 E0 F1 8A 38 A7 FD EE 5C

result PI 97 70 C7 7B 25 DB F0 24 20 A6 D6 06 44 4B 6C 9C

L (15) E6 57 1C 2A 30 20 05 BE 73 AD 91 31 C2 40 CE E3

result L E6 57 1C 2A 30 20 05 BE 73 AD 91 31 C2 40 CE E3

CYPHER TEXT E6 57 1C 2A 30 20 05 BE 73 AD 91 31 C2 40 CE E3

Тема. Функции хеширования: назначение и основные требования.

- Функции хеширования SHA-3 и GOST R 34.11-2012 (Streebog): основные параметры, структура и принцип действия преобразующей функции.
- Методы строгой аутентификации. Стандарт X.509.
- Протоколы аутентификации с симметричными алгоритмами шифрования.
- Протокол аутентификации и распределения ключей Нидхэма-Шредера.

Идеальный вариант хэш-функции – «Случайный оракул».

Случайный оракул — это абстрактный черный ящик, обладающий бесконечной памятью и работающий по следующему алгоритму:

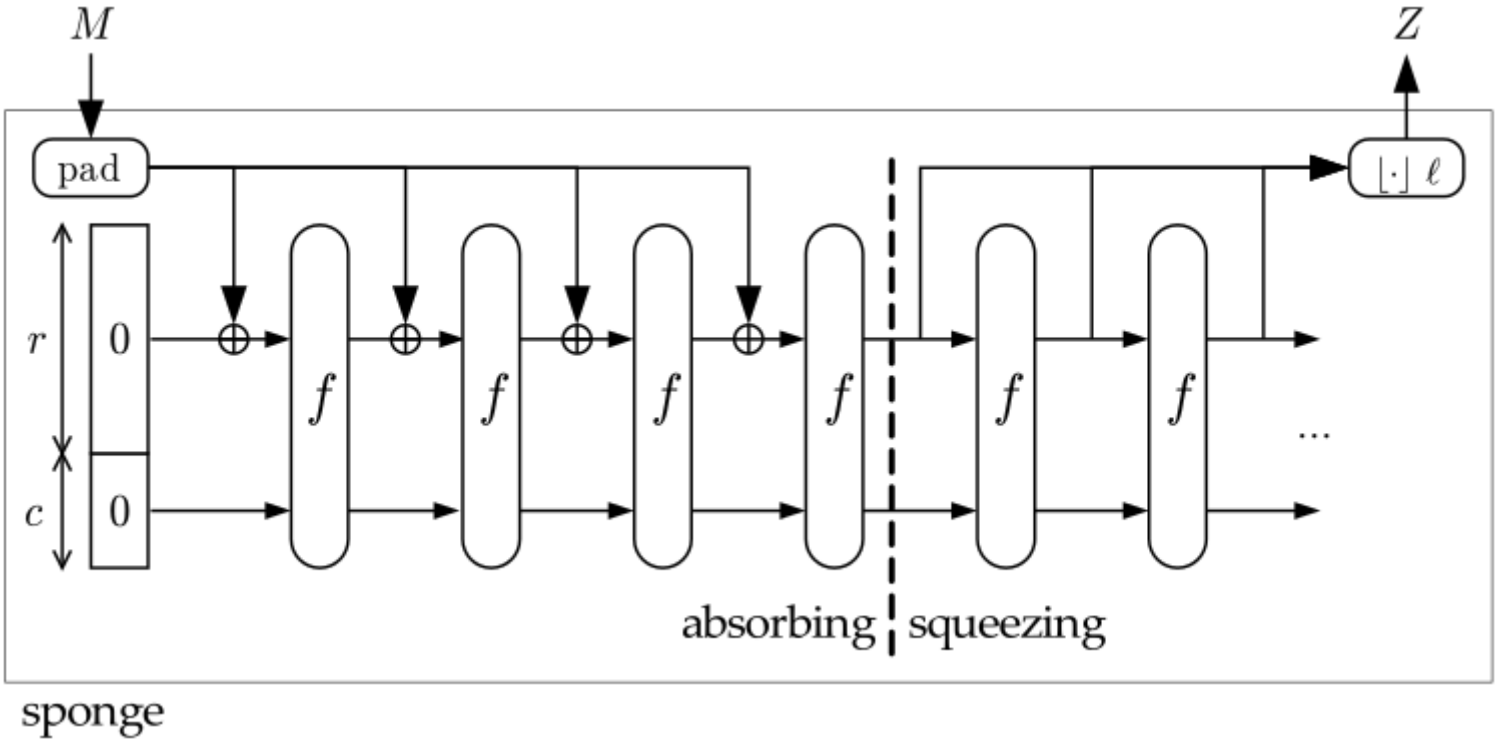
- Получает на вход строку и смотрит, работал ли он с ней ранее.
- Если нет, то генерирует истинно случайное число и сохраняет в себе пару строка-число.
- Если да, то выдает ранее сохраненное число для этой строки.

Связь между хэшируемой строкой и результатом вычислить в принципе невозможно.

Функции хеширования SHA-3 (FIPS 202, https://keccak.team)

(Авторы: Guido Bertoni, Joan Daemen, Michaël Peeters and Gilles Van Assche)

Основа SHA-3 (Keccak, «губка») – псевдослучайные перестановки, в которых коллизий не может быть по определению. Два этапа – впитывание и выжимка (отжатие).



Авторы Keccak доказали, что стойкость предложенной конструкции неотличима от случайного оракула с размером «ответа», равным c/2

Возможные варианты:

SHA-224: r=1156, c=448

SHA-256: r=1088, c=512

SHA-384: r=832, c=768

SHA-512: r=576, c=1024

Количество раундов вычисляется по формуле

$$\mathbf{n = 12 + 2\,l,}$$

где 2 l=(b/25).

Так для b=1600, Количество раундов равно 24.

Исходный код:

https://github.com/gvanas/KeccakCodePackage/blob/master/Standalone/CompactFIPS202/TweetableFIPS202.c

Доступное описание: https://ru.wikipedia.org/wiki/SHA-3 , https://habrahabr.ru/post/168707/

Демо:

http://celan.informatik.uni-oldenburg.de/kryptos/info/keccak/overview/

Тема. Асимметричные криптографические системы:

- Основные требования, односторонние функции и функции-ловушки.
- Криптографический протокол Диффи-Хеллмана.
- Асимметричные криптографические системы Эль Гамаля и Ривеста-Шамира-Адлемана (RSA).
- Электронные цифровые подписи: основные требования, алгоритмы электронной цифровой подписи.
- Цифровые подписи, основанные на асимметричных криптографических алгоритмах.
- Цифровые сертификаты. Использование сертификатов для управления криптографическими ключами.

Асимметричные криптографические системы: Основные требования, односторонние функции и функции-ловушки.

Причина появления асимметричных (несимметричных) криптоалгоритмов связана с попыткой устранения недостатка симметричных криптоалгоритмов – проблемы распределения секретных симметричных ключей.

Основа асимметричных криптоалгоритмов – применение односторонних функций с секретом и применение ключа, состоящего из двух частей: **открытый** (публичный) и **закрытый** (приватный) ключи.

Стойкость асимметричных криптоалгоритмов основана на предположении, что вычислить значение односторонней функции легко (полиномиальное время работы), а обратная задача (определение по результату исходного значения) является *трудновычисляемой*.

Так как требуется обеспечить возможность расшифрования информации, то в асимметричных криптоалгоритмах используют односторонние функции с секретом или функции-ловушки, для которых обратное преобразование является *легковычисляемым* при знании некоторого секрета. (В отличии от действительно односторонних хэш-функций).

Односторонние функции с секретом основаны на математически сложных задачах – проблема

разложения числа на простые сомножители (задача факторизации больших чисел), проблема вычисления дискретного логарифма в конечных полях и др.

Первая публикация, посвящённая криптографии с открытым ключом – статья Уитфилда Диффи и Мартина Хеллмана «Новые направления в криптографии» (New Directions in Cryptography), опубликованная в 1976 г. В последствии был разработан алгоритм Диффи-Хеллмана, позволяющий двум сторонам распределить по незащищённым каналам связи *общий секретный ключ* (который потом используется в симметричных криптоалгоритмах).

Для асимметричных криптоалгоритмов, позволяющих шифровать и расшифровывать информацию, при шифровании используется **открытый ключ**, но для расшифрования требуется **закрытый ключ**. Без знания закрытого ключа попытка расшифрования сталкивается с решением математически сложной задачи.

Криптографический протокол Диффи-Хеллмана.

Предназначен для распределения по незащищённым каналам связи между двумя сторонами A и B *общего секретного ключа K* без обмена секретной информацией. Предполагается, что злоумышленник может наблюдать за процессом передачи, но не может изменять передаваемые значения.

Работа алгоритма:

Участник Алиса

1. Генерирует случайное натуральное число *a* – закрытый ключ.

2. Генерирует открытые параметры *p* и *g*, передаёт их участнику **Бобу**.

p – простое случайное число и *g* – является первообразным корнем по модулю *p* (также является простым числом).

3. Вычисляет открытый ключ $A = g^a \bmod p$, отправляет значение *A* Бобу и принимает от него *B*.

4. Вычисляет секретный ключ $K = B^a \bmod p = g^{a \cdot b} \bmod p$ используя открытый ключ *B* и свой закрытый ключ *a*.

Участник А	**Участник В**	
1. a = **3**	1. b = 5	закрытый ключ
2. p = 11 и g = 7	2. p = 11 и g = 7	открытые параметры
3. A = 7^3 mod 11 = 2	3. B = 7^5 mod 11 = **10**	открытый ключ
4. K = **10^3** mod 11 = 10	4. K = 2^5 mod 11 = 10	секретный ключ

Так как злоумышленник не знает параметры *a* и *b*, то для него задача вычисления K является трудно вычислимой (математически сложная задача вычисления дискретного логарифма для больших целых чисел).

Асимметричные криптографические системы Ривеста-Шамира-Адлемана (RSA) и Эль-Гамаля.

Криптографический алгоритм RSA (Rivest, Shamir и Adleman) был предложен в 1977 г. Основной его стокойсти является применение математически сложной задачи разложения больших чисел на простые сомножители (факторизация).

Шифрование: $c = m^e \bmod n$

Расшифрование: $m = c^d \bmod n$

Где:

m – открытый текст;

c – зашифрованный текст;

e, n – открытый ключ (пара чисел);

d, n – закрытый ключ (пара чисел).

Пример: m = 8, e = 17, n = 77, d = 53; открытый ключ = {17, 77}; закрытый ключ = {53, 77}

$$c = 8^{17} \bmod 77 = 57$$

$$m = 57^{53} \bmod 77 = 8$$

Генерация ключей RSA:

1. Выбираются два больших простых числа p и q. (разрядность чисел напрямую влияет на стойкость).

2. Вычисляется модуль системы $n = pq$ и $\varphi(n) = (p – 1)(q – 1)$.

3. Выбирается большое число e, взаимно простое с $\varphi(n)$ и удовлетворяющее условию $1 < e < \varphi(n)$.

4. Вычисляется большое целое число d (используя расширенный алгоритм Евклида), удовлетворяющее условию:

$ed = 1 \pmod{\varphi(n)}$;

$1 < d < \varphi(n)$.

(таким образом, число d – мультипликативно обратное к числу e по модулю $\varphi(n)$)

В примере выше были выбраны:

1. $p = 7$ и $q = 11$.

2. $n = 7*11 = 77$, $\varphi(n) = (7 – 1)(11 – 1) = 60$

3. $e = 17$

4. $d = 53$

5. $(17*53) \bmod 60 = 1$

Стойкая разрядность чисел для RSA – 4096 бит и выше.

Электронные цифровые подписи: основные требования, алгоритмы электронной цифровой подписи. Цифровые подписи, основанные на асимметричных криптографических алгоритмах.

Обобщённая схема электронной цифровой подписи (ЭЦП):

Допустим, участник А хочет подписать электронный документ (ЭД), а участник В – проверить подпись. Для формирования подписи участник А вырабатывает пару {открытый ключ, закрытый ключ}, получает цифровой сертификат для открытого ключа.

1. Участник А вычисляет хэш-значение от электронного документа.
2. Шифрует полученное хэш-значение с использованием закрытого ключа – получает ЭЦП.
3. Отправляет ЭД, ЭЦП и цифровой сертификат участнику В.
4. Участник В извлекает из цифрового сертификата открытый ключ участника А.
5. Извлекает хэш-значение ЭД из цифровой подписи, используя сертифицированный открытый ключ.
6. Повторно вычисляет хэш-значение от электронного документа и сравнивает с расшифрованным.

Пример ЭЦП с использование SHA1 и RSA:

Хэш-значение от электронного документа (SHA-1):

49 FA 10 9B ED D4 72 BF BB 0D 57 FC CD 52 24 2F 28 6D EC BC

Параметры RSA:

p = 207387764168660076092594540056989467110768557l

q = 151106065952391835104579721708598232416351076l

lenght = 304 bit

n = 313375491701886337157785106120891688537062646214817123806496659281154367801850938466292953l

φ(n) = 313375491701886337157785106120891688537062645856323293685444748083980106036263238939173320 0

e = 65537

d = 754928096035732714504342776666l

{d, n} – закрытый ключ;

{e, n} – открытый ключ.

Пример цифрового сертификата:

Version: 2 (X.509v3-1996)

SubjectName: CN=Sergey Polikarpov [1607010986], DC=cryptool, DC=org

IssuerName: CN=CrypTool CA 2, DC=cryptool, DC=org

SerialNumber: 03

Validity - NotBefore: Thu Dec 03 18:56:50 2020 (201203155650Z)

NotAfter: Fri Dec 03 18:56:50 2021 (211203155650Z)

Public Key Fingerprint: 9F0E CF90 4F7E 7752 7DEF 8A57 7AD3 3C4F

SubjectKey: Algorithm rsa (OID 2.5.8.1.1), Keysize = 512

Public modulus (no. of bits = 301):

0 189D3E64 0282E893 7FAE76F2 8F523AF6

10 AB4E3754 0C0E48B4 8D8D8679 4D8780FE

20 B7238510 507B

Public exponent (no. of bits = 17):

0 010001

Certificate extensions:

Private extensions:

OID 2.206.5.4.3.2:

PrintableString:

|C:\users\serg\Application Data\C|

|rypTool\PSE/[Polikarpov][Sergey]|

|[RSA-304][1607010986].pse |

Signature: Algorithm sha1WithRSASignature (OID 1.3.14.3.2.29), NULL

0 5ECCA9BD EFFB181E 5EFBE926 DDA791EE

10 CB97CF13 FC7E984B A57C923A 73D40EF2

20 418461B9 8D2DDD3E FAD0B5CF CE93B4A7

30 E284FD69 516BEFA2 A96380B0 0736018E

40 CC63A809 84F966A3 0438AD9A AAED071A

50 A5B071BA BFE1E761 116385CA E8D82152

60 4894746A 68405811 95F36CF7 490C9160

70 0918E2EB 096E6146 2A8D79B3 24C0859D

80 A12A8C1C 4EC00CBE AD20438A 8F90AEAE

90 FFEC5E9A A60AB00C 97D4FB2C 4AF0ECFE

A0 CADB9BF6 32E937A7 B1C55CE3 EA03CFC4

B0 C41C7E76 6503BD4A AD2E9506 01DD3370

C0 D092E043 2CDDC9B6 FD7AE63C 9FA0E8DF

D0 0FEA2A94 21354A6A 7CC2159D A32B4084

E0 63500966 56C74DC4 E62C38EB EDEE813B

F0 842FC15B B5486487 E0C101A5 F3196D86

Certificate Fingerprint (MD5): 87:F8:9A:C1:EF:FD:0F:5E:12:AD:6D:5C:A9:C9:B4:BB

Certificate Fingerprint (SHA-1): F910 AF92 A3F1 2D51 7730 AB0F 7573 D333 6CCF 1369

Зашифрованное с помощью RSA хэш-значение ЭД:

Padding string: **01 00**

Algorithm ID: **30 21 30 09 06 05 2B 0E 03 02 1A 05 00 04 14**

Hash value: **49 FA 10 9B ED D4 72 BF BB 0D 57 FC CD 52 24 2F 28 6D EC BC**

ASN-1 hash value: **01 00 30 21 30 09 06 05 2B 0E 03 02 1A 05 00 04 14 49 FA 10 9B ED D4 72 BF BB 0D 57 FC CD 52 24 2F 28 6D EC BC**

Length in bits: 296

Encrypted hash value: **04 3A F8 B4 44 30 C3 42 96 9B 74 E9 2D 52 7E 24 7B 2F CB 3F 9B 6F 3E 8F 48 83 35 61 CF 87 A7 37 67 60 FA 5F A8 FF**

Length in bits: 304

Перечень основных алгоритмов электронной цифровой подписи:

1. Российские стандарты электронной цифровой подписи **ГОСТ Р 34.10-2012** (основан на сложности вычисления дискретного логарифма в группе точек эллиптической кривой).
2. Американские стандарты электронной цифровой подписи: DSA, ECDSA (DSA на основе аппарата эллиптических кривых).
3. Схемы стандарта PKCS#1, основанные на алгоритме RSA.
4. Схема Эль-Гамаля.

Цифровые сертификаты. Использование сертификатов для управления криптографическими ключами.

ПРИЧИНЫ НЕНАДЕЖНОСТИ КРИПТОГРАФИЧЕСКИХ СИСТЕМ

1. Применение нестойких криптоалгоритмов.
2. Ошибки в реализации криптоалгоритмов.
3. Неправильное применение криптоалгоритмов.
4. Человеческий фактор.

ПРИЧИНЫ НЕНАДЕЖНОСТИ КРИПТОГРАФИЧЕСКИХ СИСТЕМ

1. Применение нестойких криптоалгоритмов.

- Использование собственных (самодельных) криптоалгоритмов
  - уязвимая хеш-функция NTLM
- Использование скомпрометированных криптоалгоритмов (по незнанию или для совместимости с устаревшими системами)
  - A5/2
  - RC4
  - md5
  - SHA1

2. Ошибки в реализации криптоалгоритмов.

- наличие программных ошибок (уязвимостей)
  - попадание ключевой информации в непредусмотренные места (файл подкачки, общедоступные области памяти, общедоступный кэш процессора и др.)
  - отсутствие защиты от преднамеренного вмешательства в работу программы (heartlich)
- наличие внедрённых в криптоалгоритм закладок

- SkipJack
- RSALab

- наличие зависимости времени выполнения шифрования от значения ключей
  - пример «звучащих» конденсаторов для асимметричного криптоалгоритма
  - особенности работы GPU реализации

3. Неправильное применение криптоалгоритмов.

- малая длина ключа
  - если ключ зависит от слабого пароля
  - игнорирование правил генерации ключей и векторов инициализации
    - пример с SHA1.
  - недостаточная энтропия источника случайных чисел, используемого для генерации ключей
    - скандал с RSAlab
    - замена источника энтропии в Linux и FreeBSD
- неправильный выбор криптоалгоритма
- неправильный режим работы криптоалгоритма
  - применение режима ECB (электронной кодовой книги) для блочного шифра при шифровании

повторяющихся областей структурированных файлов (графика, документы, таблицы и т.д.).

- повторное наложение одинаковой гаммы поточного шифра (без применения вектора инициализации или соли).

- хранение ключевых данных вместе с данными
  - продукты компании Microsoft:
    - Microsoft Access (ранние версии) – хранение ключа в заголовке базы данных
  - android

4. Человеческий фактор.

1. Применение нестойких криптоалгоритмов.

- Использование собственных (самодельных) криптоалгоритмов
  - уязвимая хеш-функция NTLM
- Использование скомпрометированных криптоалгоритмов (по незнанию или для совместимости с устаревшими системами)

> **OpenNET.ru (27.06.2015 09:23):**
> Комитет IETF (Internet Engineering Task Force), занимающийся развитием протоколов и архитектуры интернета, выпустил RFC-7568, переводящий SSLv3 в разряд устаревших протоколов и предупреждающий, что его применение представляет угрозу безопасности систем. Опубликованный RFC требует прекратить применение SSLv3 и запрещает отправку клиентами и серверами пакетов ClientHello/ServerHello с полями ClientHello.client_version/ ServerHello.server_version со значением {03,00}, а в случае получения подобных пакетов регламентирует необходимость разорвать соединение. Вместо SSLv3 рекомендовано использовать TLSv1.2 (RFC 5246).
> Напомним, что SSLv3 был скомпрометирован осенью прошлого года, после выявления атаки POODLE (CVE-2014-3566), которая позволяет атакующему извлечь из зашифрованного канала связи закрытую информацию, такую как содержимое Cookies, которые могут содержать идентификаторы сеанса и коды доступа, что сводит на нет средства обеспечения безопасного соединения на основе протокола SSL 3.0. С тех пор поддержка SSLv3 была отключена по умолчанию в браузерах и некоторых серверных системах, но оставалась проблема с обеспечением совместимости с устаревшими системами и возможность совершения атак, вызванных откатом соединения на SSLv3. Официальный запрет применения SSLv3 на уровне интернет-стандарта призван решить данные проблемы и стимулировать полное исключение SSLv3 из обихода.

  - SHA1

OpenNET.ru (11.10.2015 22:13) До конца года ожидается появление практических атак по подбору коллизий для SHA-1:

Брюс Шнайер, известный эксперт в области компьютерной безопасности, сообщил, что данный им три года назад прогноз стойкости алгоритма хэширования SHA-1 оказался излишне оптимистичным и появление первой практической атаки по подбору коллизий для SHA-1 можно ждать не в 2018 году, а до конца текущего года. Напомним, что по прогнозу Шнайера в 2012 году затраты на подбор коллизии в SHA-1 оценивались в 2 млн долларов, в 2015 году прогнозировалось уменьшение стоимости до 700 тысяч, к 2018 году до 173 тысяч, а к 2021 до 43 тысяч долларов.

Группа исследователей из научных учреждений Голландии, Франции и Сингапура разработала оптимизированный метод подбора коллизий для функции сжатия, используемой в SHA-1 (не сам алгоритм SHA-1), который существенно сокращает время атаки и стоимость её проведения. При проведении эксперимента представленный алгоритм позволил осуществить подбор префикса за 9-10 дней на кластере из 64 GPU. При этом, стоимость вычислений, с учётом создания такого кластера на базе вычислительной мощности на Amazon EC2, составила всего 2 тысячи долларов. Время подбора реальной коллизии для произвольного хэша SHA-1 оценивается в 49 до 78 дней при вычислениях на кластере из 512 GPU, стоимость работы которого на базе Amazon EC2 составит 75-120 тысяч долларов.

С учётом того, что работающие атаки могут стать реальностью в ближайшие несколько месяцев исследователи рекомендуют пересмотреть сроки перевода SHA-1 в разряд устаревших технологий. Принятый ранее план предусматривает отказ от SHA-1 начиная с 2017 года, в то время как исследователи безопасности настаивают, что с учётом увеличения эффективности проведения атак, SHA-1 должен прекратить своё существование уже в январе 2016 года. Ситуацию усугубляет то, что около 28% сайтов в сети пользуются SHA-1 для заверения своих HTTPS-сертификатов и отрасль оказалась не готова к экстренному отказу от SHA-1.

OpenNET.ru (04.03.2015 13:56) Новая атака на SSL/TLS, позволяющая организовать перехват HTTPS-трафика:

Исследователи из французского института INRIA выявили новый вид атаки на SSL/TLS, который получил название FREAK, по аналогии с ранее выявленными атаками POODLE, BREACH, CRIME и BEAST. Суть атаки сводится к инициированию отката соединения на использование разрешённого для экспорта набора шифров, включающего недостаточно защищённые устаревшие алгоритмы шифрования. Проблема позволяет вклиниться в соединение и организовать анализ трафика в рамках защищённого канала связи, используя уязвимость (CVE-2015-0204), выявленную во многих SSL-клиентах и позволяющую сменить шифры RSA на RSA_EXPORT и выполнить дешифровку трафика, воспользовавшись слабым эфемерным ключом RSA.

Предоставляемый в RSA_EXPORT 512-битный ключ RSA уже давно не применяется в серверном и клиентском ПО, так как считается небезопасным и подверженным атакам по подбору. Для подбора ключа RSA-512 исследователям потребовалось около семи с половиной часов при запуске вычислений в окружении Amazon EC2. Ключ достаточно подобрать для каждого сервера один раз, после чего подобранный ключ может использоваться для перехвата любых соединений с данным сервером (mod_ssl по умолчанию при запуске сервера генерирует один экспортный RSA-ключ и повторно использует его для всех соединений).

На стороне клиента уязвимость затрагивает OpenSSL (исправлено в 0.9.8zd, 1.0.0p и 1.0.1k), браузер Safari и разнообразные встраиваемые и мобильные системы, включая Google Android и Apple iOS. Что касается серверов, то сканирование сети показало, что набор RSA_EXPORT поддерживается приблизительно на 36.7% из общей массы сайтов и на 9.7% из миллиона крупнейших сайтов. Для защиты сервера на базе Apache к параметрам директивы SSLCipherSuite следует добавить "!EXPORT". Протестировать сайт на наличие уязвимости можно на данной странице.

Дополнение: Проблеме также оказались подвержены клиентские и серверные выпуски ОС Windows.

2. Ошибки в реализации криптоалгоритмов.

- наличие программных ошибок (уязвимостей)
  - попадание ключевой информации в непредусмотренные места (файл подкачки, общедоступные области памяти, общедоступный кэш процессора и др.)

habrahabr.ru/post/169717/ (17 февр. 2013 г.)

Морозная атака на шифрование в Android

OpenNET.ru (26.07.2013 10:01) Обновление GnuPG с устранением уязвимости, позволяющей восстановить закрытые RSA-ключи

Представлено корректирующее обновление пакета GnuPG 1.4.14 (GNU Privacy Guard) и библиотеки Libgcrypt 1.5.3 с реализацией компонентов, лежащих в основе механизмов шифрования, применяемых в GnuPG 2.0. В указанных выпусках устранена интересная уязвимость, позволяющая восстановить содержимое закрытого RSA-ключа другого пользователя многопользовательской системы, используя особенности помещения данных в совместно используемый L3-кэш.

Суть метода сводится к определению содержимого кэша, полагаясь на различие времени доступа к прокэшированным и отсутствующим в кэше значениям. Атакующий добивается вытеснения участков кэша своими данными с последующим выявлением попадания в кэш сторонних данных на основе методов статистического анализа. В качестве эталона используется установленный в системе вариант программы gpg, который одинаков для атакующего и жертвы. При запуске нескольких копий, программы размещаются в памяти с использованием разделяемой памяти, с обработкой изменяемых данных в отдельных страницах, но с попаданием в кэш сходным образом (зная как размещена программа в памяти, атакующий может на основе оценки изменения времени доступа попытаться подобрать данные из не разделяемых страниц памяти).

Представленная техника атаки позволяет восстановить более 98% битов закрытого ключа при проведении в системе каждого цикла дешифровки данных или формирования цифровой подписи. В отличие от ранее известных атак, представленный метод не требует привязки выполнения на одном ядре CPU шифрующего кода и кода злоумышленника, так как L3-кэш разделяется между всеми процессорными ядрами. Более того, атака может применяться и в виртуализированных окружениях, в которых злоумышленник может атаковать GnuPG-сеанс, выполняемый в другой гостевой системе.

Несмотря на то, что отдельного корректирующего обновления GnuPG 2.0 не выпущено, проблема также затрагивает ветку GnuPG 2.0.x, для пользователей которой выпущено обновление Libgcrypt 1.5.3 (в ветке GnuPG 2.0.x базовая логика шифрования вынесена в отдельную библиотеку).

- отсутствие защиты от преднамеренного вмешательства в работу программы

OpenNET.ru (08.04.2014 10:49) В OpenSSL обнаружена критическая уязвимость, которая может привести к утечке закрытых ключей:

В OpenSSL выявлена одна из самых серьёзных уязвимостей (CVE-2014-0160) в истории проекта, затрагивающая огромное число сайтов, серверных систем и клиентских приложений, использующих OpenSSL 1.0.1 для организации обмена данными через защищённое соединение. Суть проблемы проявляется в возможности доступа к 64Кб памяти клиентского или серверного процесса с которым установлено шифрованное соединение.

Практическая опасность уязвимости связана с тем, что в подверженной утечке области памяти могут размещаться закрытые ключи или пароли доступа, которые потенциально могут быть извлечены удалённым злоумышленником. При удачном проведении атаки злоумышленник может получить содержимое ключа, используемого для организации шифрованного доступа к серверу, и организовать перехват на транзитном узле защищённого трафика. Также не исключена утечка паролей, идентификаторов сессий и других закрытых данных клиентских приложений при попытке их соединения с подконтрольными злоумышленникам серверными системами.

Причиной проблемы является отсутствие проверки выхода за допустимые границы в коде реализации TLS-расширения heartbeat (RFC 6520), что позволяет инициировать отправку удалённой стороне до 64Кб данных из области за границей текущего буфера. При этом возможна отправка произвольных 64Кб данных, а не только примыкающих к границе буфера (путем повторения запросов атакующий может шаг за шагом прочитать всё содержимое памяти процесса). По некоторым оценкам проблема охватывает до половины поддерживающих защищённое соединение серверных систем в Сети, включая собранные с OpenSSL 1.0.1 web-серверы (Apache httpd, nginx), почтовые серверы, XMPP-серверы, VPN-системы, шлюзы и скрытые сервисы анонимной сети Tor.

Связанная с атакой активность не проявляется в серверных логах, что затрудняет выявление фактов эксплуатации уязвимости. Возможность создания рабочего эксплоита для извлечения ключей безопасности без оставления следов в логе подтверждена исследователями из компаний Google и Codenomicon, выявивших уязвимость. В связи с тем, что проблема присутствует в коде OpenSSL уже более двух лет и, при этом, невозможно отследить по логу уже совершённые атаки, помимо установки обновления пользователям рекомендуется поменять SSL-сертификаты, ключи доступа и пароли. Для выявления возможного применения атаки в диком виде продвигается инициатива по развёртыванию сети подставных honeypot-серверов, фиксирующих попытки эксплуатации уязвимости.

Проблема проявляется только в актуальной стабильной ветке OpenSSL 1.0.1 и тестовых выпусках 1.0.2, прошлые стабильные ветки 0.9.x и 1.0.0x уязвимости не подвержены. Системы, использующие выпуски OpenSSL 1.0.1[abcdef], требуют срочного обновления. Уязвимость устранена в выпуске OpenSSL 1.0.1g. Обновления пакетов уже выпущены для RHEL 6, CentOS 6, Fedora 19/21, FreeBSD 8.4+, Ubuntu 12.04-13.10, Debian wheezy/jessie/sid, ALT Linux, CRUX, Mageia, CRUX, OpenBSD 5.3+. Проблема пока не исправлена в OpenSUSE 12.2+, NetBSD 5.0+. SUSE Linux Enterprise Server и Debian Squeeze проблеме не подвержены. В качестве запасного варианта отмечается возможность пересборки OpenSSL с опцией "-DOPENSSL_NO_HEARTBEATS", отключающей TLS-расширение heartbeat. Для проверки серверных систем на предмет наличия уязвимости подготовлены специальные online-сервисы.

OpenNET.ru (10.04.2014 13:03) Heartbleed-уязвимость в OpenSSL могла эксплуатироваться с ноября прошлого года :

Стали появляться свидетельства возможного применения Heartbleed-уязвимости в OpenSSL (CVE-2014-0160) для совершения вредоносных действий за несколько месяцев до выявления проблемы сотрудниками компаний Google и Codenomicon. Следы одной из таких атак зафиксированы компанией MediaMonks в журналах аудита, датированных ноябрём прошлого года. Сохранённые в журналах пакеты с нескольких серверов, подозреваемых во вредоносной активности, совпали по своему характеру с пакетами, применяемыми при эксплуатации Heartbleed-уязвимости.

По мнению Брюса Шнайера, известного эксперта в области компьютерной безопасности, Heartbeat-уязвимость в OpenSSL следует причислить к категории катастрофических уязвимостей, уровень опасности которой составляет 11 баллов, если рассматривать существующую 10-бальную шкалу степеней опасности с учётом того, что OpenSSL является самой распространённой криптографической библиотекой в Сети.

Благодаря широкому освещению проблемы за два дня с момента её обнародования около 1/3 всех серверов уже применили обновление с устранением уязвимости. Тем не менее, по предварительным данным в Сети ещё остаются уязвимыми около 600 тысяч серверов. Но проблема далека от своего решения - непонятно, что делать со встраиваемыми и мобильными продуктами, подверженными уязвимости, но не предусматривающими возможность автоматического обновления прошивки.

Кроме того, начинается волна атак на клиентские приложения, использующие OpenSSL. Например, вслед за появлением эксплоитов для серверных систем уже доступен прототип эксплоита с реализацией фиктивного HTTPS-сервера, при обращении к которому осуществляется атака на клиента. Эксплоит успешно протестирован для извлечения данных из памяти таких приложений, как wget 1.15, curl 7.36.0, links 2.8, git 1.9.1, MariaDB 5.5.36 (клиент), nginx 1.4.7 (в режиме прокси). Например, можно извлечь параметры прошлых запросов, в том числе содержащих пароли доступа.

В условиях возможности незаметного проведения атаки, без оставления следов в логе, также предстоит длительный процесс смены SSL-сертификатов, ключей шифрования и обычных паролей, отсутствие утечки которых невозможно гарантировать. Потенциально любой пароль и сертификат мог попасть в руки злоумышленников, и непонятно, когда и где подобные утечки могут проявиться. Из крупных сайтов, которые потенциально могли подвергнуться атаке отмечаются Twitter, GitHub, Yahoo, Tumblr, Steam, DropBox, а также многие банки и финансовые сервисы.

Шнайер считает близкой к единице вероятность того, что различные спецслужбы уже успели воспользоваться

уязвимостью для массового извлечения приватных ключей. Другой вопрос, случайно или нет подобная уязвимость появилась в OpenSSL. Даже если проблема была внесена случайно, за два года присутствия в кодовой базе заинтересованные лица вполне могли её обнаружить и молча использовать.

Дополнение: администраторам у которых сохранились дампы трафика за время до публикации информации об уязвимости, предлагается проанализировать наличие в них сигнатур "18 03 02 00 03 01 40 00" или "18 03 01 00 03 01 40 00" (вместо "40 00" в конце может быть меньшее число), свидетельствующих о применении эксплоита. Особое внимание рекомендуется уделить на подсеть 193.104.110.* с которой в ноябре была выявлена подобная активность, а также другие подсети с которых наблюдается активность ботов.

OpenNET.ru (19.01.2015 21:18) Критическая уязвимость в криптографической библиотеке PolarSSL :

В развиваемой компанией ARM свободной крипографической библиотеке PolarSSL выявлена критическая уязвимость (CVE-2015-1182), которая потенциально может привести к выполнению кода злоумышленника при обработке средствами библиотеки специально оформленных последовательностей ASN.1 из сертификатов X.509. Проблема может проявляться в серверных и клиентских приложениях, использующих PolarSSL при организации шифрованного канала связи с подконтрольным злоумышленнику клиентом или сервером.

Среди программ, поддерживающих сборку с PolarSSL, можно отметить OpenVPN-NL (уязвим и требует обновления, так как использует по умолчанию PolarSSL), OpenVPN, cURL, FreeRDP, PowerDNS. Проблема проявляется во всех выпусках PolarSSL 1.x, включая актуальные версии 1.3.9 и 1.2.12. Исправление пока доступно в виде патча.

- наличие внедрённых в криптоалгоритм закладок
  - SkipJack
  - RSALab

OpenNET.ru (01.01.2014 22:02) Опубликован прототип бэкдора в генераторе псевдослучайных чисел Dual_EC_DRBG, входившем в стандарт NIST :

Aris Adamantiadis, исследователь безопасности из Бельгии, развивающий проект libssh, опубликовал рабочий прототип приложения, подтверждающего теорию о возможном наличии бэкдора в алгоритме генерации псевдослучайных чисел Dual EC DRBG, до недавних пор входящим в стандарт NIST SP 800-90 и использованном по умолчанию в продуктах Bsafe и RSA Data Protection Manager от компании RSA.

Алгоритм Dual EC DRBG, описывающий способ генерации псевдослучайных чисел на основе методов криптографии по эллиптическим кривым, был разработан и продвинут в состав стандарта Агентством национальной безопасности США (АНБ). Теоретические опасения о возможных проблемах в Dual_EC_DRBG были опубликованы ещё в 2007 году, но всерьёз они насторожили общественность лишь в сентябре 2013 года, после публикации Эдвардом Сноуденом материалов, свидетельствующих о работе АНБ по внедрению бэкдора, кардинально упрощающего предсказание генерируемых через Dual_EC_DRBG последовательностей.

После появления этих сведений компания RSA и Национальный институт стандартов и технологий США (NIST) выпустили рекомендации не использовать Dual_EC_DRBG. В декабре появились новые материалы, указывающие на заключение секретного контракта между АНБ и RSA, размером в 10 млн долларов, подразумевающего применение Dual_EC_DRBG в качестве алгоритма по умолчанию в Bsafe.

Суть проблемы в Dual_EC_DRBG связана с наличием в рекомендованной стандартом эталонной реализации

алгоритма нескольких констант, которые потенциально могут быть связаны с наличием скрытого решения, известного только тем, кто знает параметры генерации данных констант. Подобное скрытое решение не исключает возможность определения состояния генератора случайных чисел, достаточного для предсказания выводимой случайной последовательности, при наличии данных об уже сгенерированных 32 случайных байтов.

Aris Adamantiadis сумел успешно продемонстрировать возможность предсказывать выдаваемые на выходе значения, используя лишь одну изменённую константу в Dual_EC_DRBG. Весь код и инструкции, позволяющие повторить эксперимент, опубликованы на GitHub. Параметры генерации констант, указанных в стандарте NIST, остаются известны только АНБ. При этом данные константы обязательны для использования в неизменном виде при прохождении сертификатции по FIPS 140-2.

- наличие зависимости времени выполнения шифрования от значения ключей
  - пример «звучащих» конденсаторов для асимметричного криптоалгоритма
  - особенности работы GPU реализации

3. Неправильное применение криптоалгоритмов.

- малая длина ключа
  - если ключ зависит от слабого пароля

    Анализ утёкших паролей Gmail, Yandex и Mail.Ru (habrahabr.ru/post/236759/ от 15 сент. 2014 г.)
  - игнорирование правил генерации ключей и векторов инициализации
    - пример с SHA1 и протокол Диффи-Хелмана

Security Week 43: непростые простые числа, криптография в HDD, патчи Adobe и Oracle (http://habrahabr.ru/company/kaspersky/)

- недостаточная энтропия источника случайных чисел, используемого для генерации ключей
  - скандал с RSAlab
  - замена источника энтропии в Linux и FreeBSD

OpenNET.ru (19.11.2013 09:44) Для ядра Linux 3.13 представлены патчи с улучшением генерации случайных чисел :

Разработчик Теодор Тсо представил патч, улучшающий работу со случайными числами для ядра 3.13. Наиболее заметными изменениями являются улучшение производительности и повышение качества энтропии, а также ряд улучшений, касающихся работы генератора случайных чисел на платформах, отличных от x86.

Например, как один из источников энтропии теперь может использоваться регистр времени, который слишком груб для точного отслеживания времени, однако годится в качестве одного из источников энтропии. Кроме этого реализован режим "канарейки", при котором в лог ядра (printk) выводится сообщение, если программа попытается использовать /dev/urandom до того как он будет полностью инициализирован, что может потенциально привести к проблемам с надёжностью криптографических операций.

На платформе x86 данная проблема как правило не возникает (у самого разработчика на ноутбуке инициализация этого устройства завершается на 1.5 секунды раньше чем монтируется rootfs и какая-либо программа получает шанс использовать устройство), однако предполагается, что это может быть проблемой на платформах ARM или MIPS.

В будущем предполагается реализовать поведение при котором при недостаточном накоплении энтропии выполнение программы приостанавливается до момента когда устройство /dev/urandom будет

инициализировано и готово вернуть программе качественные случайные данные, что должно положительно сказаться на безопасности.

- неправильный выбор криптоалгоритма
- неправильный режим работы криптоалгоритма
  - применение режима ECB (электронной кодовой книги) для блочного шифра при шифровании повторяющихся областей структурированных файлов (графика, документы, таблицы и т.д.).
  - повторное наложение одинаковой гаммы поточного шифра (без применения вектора инициализации или соли).
- хранение ключевых данных вместе с данными
  - продукты компании Microsoft:
    - Microsoft Access (ранние версии) – хранение ключа в заголовке базы данных
  - android

    habrahabr.ru/post/254899/ (4 апр. 2015 г)

    Шифрование в NQ Vault оказалось обычным XOR-ом, и это не самое плохое.

4. Человеческий фактор.

   habrahabr.ru/post/181372/ (29 мая 2013 г.)

   «Вы опасно некомпетентны в криптографии»