Hashes

Захар Назаров

April 2024

Содержание

1 Введение **2**

2 Термины и определения **2**

3 Архитектуры хеш-функций **3**
3.1 Губка . . . 3
3.2 HAIFA . . . 4

4 MD5 **4**

5 SHA-1 **6**

6 SHA-2 (только SHA-256) **7**

7 SHA-3 **8**

8 ГОСТ Р 34.11-94 **10**

9 ГОСТ 34.11-2018 (Стрибог) **11**

10 BLAKE-256 **12**

11 RIPEMD-256 **15**

12 Whirlpool **16**
12.1 Криптоанализ. Rebound attack . . . 17
12.1.1 Этап предварительных вычислений . . . 18
12.1.2 Inbound phase. 1 этап . . . 19
12.1.3 Inbound phase. 2 этап . . . 19
12.1.4 Outbound phase. 3 этап . . . 19
12.1.5 Outbound phase. 4 этап . . . 19
12.1.6 Rebound атака на 5.5 раундов . . . 19
12.1.7 Rebound атака на 7.5 раундов . . . 20
12.1.8 Вывод . . . 20
12.2 Заключение . . . 20

13 Fast Syndrome Based Hash (FSB) **20**

14 Список литературы **21**

1. Введение

В данной работе описываются две архитектуры построения хеш-функций: "Sponge-конструкция"и "HAIFA а также 10 конкретных хеш-функций: MD5, SHA-1, SHA-2 (только SHA-256), SHA-3, ГОСТ Р 34.11-94, ГОСТ 34.11-2018 (Стрибог), BLAKE-256, RIPEMD-256, Whirlpool и Fast Syndrome Based Hash (FSB). Устройство Whirlpool и его особенности рассматриваются более детально, также описаны атаки на него.

2. Термины и определения

Для понимания работы требуется ознакомиться со следующими определениями:

Определение 2.1. Конструкция Меркла-Дамгарда - это схема построения криптографических хеш-функций. Внутри себя эта схема использует функцию f, которая принимает на вход число длиной $2n$, а на выходе выдает число длиной n. На вход этой конструкции подается сообщение M (длиной не более чем 2^n), и инициализирующий вектор IV длиной n. Алгоритм работы:

1) Если входное сообщение M не кратно n, то оно дополняется справа нулями, чтобы оно стало кратно n.
2) Далее сообщение дополняется справа блоком длины n, в котором записана исходная длина M, и разбивается на K блоков длины n: $x_1, ..., x_K$.
3) $s_0 = IV$ (если IV нет, то s_0 заполняется нулями).
4) Для $i = 1, ..., K$ вычислить $s_i = f(s_{i-1}, x_i)$.
5) Выдать s_K на выход.

Определение 2.2. Конструкция Миагути - Пренеля - это схема построения функции сжатия. На вход она $F(h, m)$ принимает 2 блока размера n и выдает блок размера n. Внутри себя F использует блочный шифр E. Конструкция выглядит следующим образом:
$F(h, m) = E_h(m) \oplus h \oplus m$
Если длина ключа E не совпадает с размером h или необходимо сделать преобразования ключа, то h пропускают через корректирующую функцию g, которая делает длину блока равной необходимой длине ключа или же преобразует ключ.

Определение 2.3. Метод "встречи посередине"(meet-in-the-middle attack) представляет собой класс атак на криптографические алгоритмы, которые асимптотически сокращают время, необходимое для полного перебора, используя принцип "разделяй и властвуй".

Определение 2.4. Термин дифференциального анализа "разница"(difference) означает переменную $a = x \oplus y$, где x, y - вход в криптографическую функцию F. Это называют входной разницей. Выходной разницей называется $b = F(x) \oplus F(y)$.

Определение 2.5. Термин дифференциального анализа "дифференциал"(differential) означает кортеж (a, b), где $a = x \oplus y$, $b = F(x) \oplus F(y)$, x, y - вход в функцию F (например, S-box). Как правило, исследуют вероятный дифференциал (который просто называют дифференциалом), он удовлетворяют условию $F(a) = b$.

Определение 2.6. Коллизия, полусвободная коллизия или полусвободная почти коллизия определяются следующим образом:

1) коллизия:
   IV - фиксирован.
   $f(M_t, IV) = f(M'_t, IV), M_t \neq M'_t$
2) почти коллизия:
   IV - фиксирован.
   $f_{M_t} = f(M_t, IV), f_{M'_t} = f(M'_t, IV), M_t \neq M'_t$
   небольшое число бит хешей f_{M_t} и $f_{M'_t}$ различны
3) полусвободная коллизия:
   $f(M_t, H_{t-1}) = f(M'_t, H_{t-1}), M_t \neq M'_t$

3. Архитектуры хеш-функций

Рассмотрим две основные архитектуры хеш-функции: "губка"и "HAIFA".

3.1. Губка

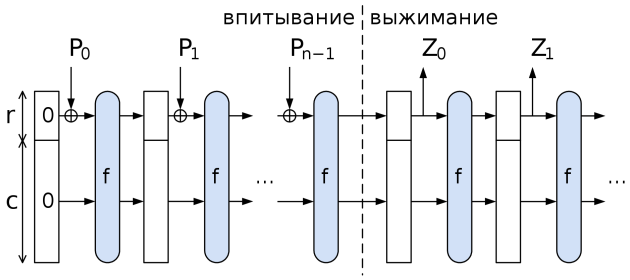

Рис. 1. Губка

Функция губки (sponge function) - это одна из архитектур, которая используется для построения хеш-функций. Губка состоит из состояния S (вектор битов) и функции псевдослучайной перестановки f. Состояние S делится на 2 последовательные части: c и r, которые состоят из $|c|$ и $|r|$ бит соответственно.

Алгоритм работы

На вход поступает сообщение M, которое разбивается на последовательные блоки длиной $|r|$. Затем эти блоки последовательно подаются в губку. Состояние S инициализируется нулями.

Этап впитывания (absorbing). На вход губке подается очередной блок текста длиной $|r|$. Он "ксорится"с частью r состояния S, затем к S применяется функция f. Затем подается следующий блок текста и аналогично обрабатывается, пока не закончатся блоки исходного сообщения.

Затем идет следующий этап. Определим желаемую длину выхода губки и создадим переменную res этой длины, заполненную нулями.

Этап выжимания (squeezing). Из r (часть S) биты копируются res, затем к S применяется f. Если переменная не заполнилась, то снова переносим данные из r в res. Заметим, что переменная res заполняется последовательно данными из r. По итогу в res должна находится конкатенация данных из r, полученных на разных этапах.

При необходимости можно взять, обрезанную часть res, если выход губки не кратен r.

Особенности, достоинства, недостатки.

Губка позволяет получить выход любой длины, задавая количество шагов в фазе выжимания. Также можно менять размер c и r состояния S. Увеличивая размер c, мы делаем выходную последовательность более устойчивой к криптоанализу и наоборот. Увеличивая размер r, мы увеличиваем скорость выдачи результирующих битов и наоборот. Поэтому важно найти баланс между стойкостью и "скоростью"губки.

Если злоумышленник знает представление функции f в виде циклов полностью или частично, то он способен строить коллизии для хеш-функции, в основе которой лежит губка [1].

3.2. HAIFA

Конструкция для хеш-функции HAIFA (HAsh Iterative FrAmework) является модификацией классической конструкции Меркла—Дамгарда.

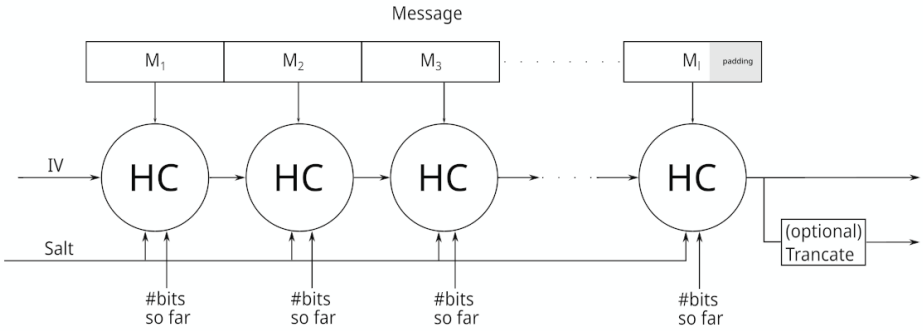

Рис. 2. HAIFA

Алгоритм работы

Перед хешированием сообщение дополняется следующими битами: одна единица, определенное количество нулей, длина сообщения в битах, длина хеша в битах. Нулей добавляется такое количество, которое сделает длину сообщения кратной длине блока.

На вход HAIFA подается сообщение M, инициализирующий вектор IV и соль(salt):

1) Сообщение M дополняется и разбивается на блоки одинаковой длины M_i.
2) $h_0 = IV$.
3) Пока есть необработанный блок M, вычисляется функция $C : h_i = C(h_{i-1}, M_i, bits, salt)$.
4) Финальное значение h_i подается на выход.

$bits$ на каждом этапе хеширования определяется количеством захешированных битов исходного сообщения к данному моменту.

Особенности, достоинства, недостатки.

Конструкция HAIFA устойчива к коллизиям, если функция C устойчива к коллизиям. Соль пресекает атаки, которые используют предварительные вычисления. Функция C является узким местом в данной конструкции, если не удается подобрать стойкую функцию, то HAIFA также не будет стойкой.

4. MD5

MD5(Message Digest 5) — алгоритм хеширования, разработанный в MIT в 1991 году. MD5 использует конструкцию Меркла-Дамгарда. В 1992 году алгоритм стал стандартом [2], опишем его.

Функции и константы

В MD5 используются следующие функции:

1) $F(X, Y, Z) = (X \wedge Y) \vee (\neg X \wedge Z)$

2) $G(X,Y,Z) = (X \wedge Z) \vee (\neg Z \wedge Y)$

3) $H(X,Y,Z) = X \oplus Y \oplus Z$

4) $I(X,Y,Z) = Y \oplus (\neg Z \vee X)$

Также используется 64-элементный список 32-битных констант $T : T[n] = int(2^{32} * |sin(n)|), n = 1 \ldots 64$.

Алгоритм MD5

У MD5 есть свое внутреннее 128-битное состояние, которое задается 32-битными словами A,B,C,D. Оно инициализируется следующим образом:

1) $A = 0x01234567$

2) $B = 0x89abcdef$

3) $C = 0xfedcba98$

4) $D = 0x76543210$

Входное сообщение M дополняется единицей, нулями и длиной сообщения в битах (64 бита). Нулей добавляется такое количество, которое сделает длину сообщения кратной 512. Далее алгоритм по очереди обрабатывает 512-битные блоки.

Перед обработкой очередного блока 128-битное состояние запоминается, для простоты заведем переменную res, в которой будем хранить это состояние.

Обработка 512-битного блока проходит в 4 раунда, каждый раунд состоит из 16 итераций (блок делится на 16 32-битных частей и хранится в массиве X). После этого к текущему состоянию добавляется (операция "oplus") предыдущее состояние (оно у нас хранится в res).

Раунды выглядят следующим образом:

```
/* Round 1. */
/* Let [abcd k s i] denote the operation
     a = b + ((a + F(b,c,d) + X[k] + T[i]) <<< s). */
/* Do the following 16 operations. */
[ABCD  0  7  1]  [DABC  1 12  2]  [CDAB  2 17  3]  [BCDA  3 22  4]
[ABCD  4  7  5]  [DABC  5 12  6]  [CDAB  6 17  7]  [BCDA  7 22  8]
[ABCD  8  7  9]  [DABC  9 12 10]  [CDAB 10 17 11]  [BCDA 11 22 12]
[ABCD 12  7 13]  [DABC 13 12 14]  [CDAB 14 17 15]  [BCDA 15 22 16]

/* Round 2. */
/* Let [abcd k s i] denote the operation
     a = b + ((a + G(b,c,d) + X[k] + T[i]) <<< s). */
/* Do the following 16 operations. */
[ABCD  1  5 17]  [DABC  6  9 18]  [CDAB 11 14 19]  [BCDA  0 20 20]
[ABCD  5  5 21]  [DABC 10  9 22]  [CDAB 15 14 23]  [BCDA  4 20 24]
[ABCD  9  5 25]  [DABC 14  9 26]  [CDAB  3 14 27]  [BCDA  8 20 28]
[ABCD 13  5 29]  [DABC  2  9 30]  [CDAB  7 14 31]  [BCDA 12 20 32]

/* Round 3. */
/* Let [abcd k s t] denote the operation
     a = b + ((a + H(b,c,d) + X[k] + T[i]) <<< s). */
/* Do the following 16 operations. */
[ABCD  5  4 33]  [DABC  8 11 34]  [CDAB 11 16 35]  [BCDA 14 23 36]
[ABCD  1  4 37]  [DABC  4 11 38]  [CDAB  7 16 39]  [BCDA 10 23 40]
[ABCD 13  4 41]  [DABC  0 11 42]  [CDAB  3 16 43]  [BCDA  6 23 44]
[ABCD  9  4 45]  [DABC 12 11 46]  [CDAB 15 16 47]  [BCDA  2 23 48]

/* Round 4. */
/* Let [abcd k s t] denote the operation
     a = b + ((a + I(b,c,d) + X[k] + T[i]) <<< s). */
/* Do the following 16 operations. */
[ABCD  0  6 49]  [DABC  7 10 50]  [CDAB 14 15 51]  [BCDA  5 21 52]
[ABCD 12  6 53]  [DABC  3 10 54]  [CDAB 10 15 55]  [BCDA  1 21 56]
[ABCD  8  6 57]  [DABC 15 10 58]  [CDAB  6 15 59]  [BCDA 13 21 60]
[ABCD  4  6 61]  [DABC 11 10 62]  [CDAB  2 15 63]  [BCDA  9 21 64]
```

Рис. 3. Раунды MD5

Конечное 128-битное состояние является выходом MD5.

5. SHA-1

SHA-1 (Secure Hash Algorithm 1) — алгоритм хеширования, разработанный АНБ США в 1995 году [3]. Этот хеш также стал стандартом [4].

Функции и константы

В SHA-1 используются следующие функции и константы:

1) $F_t(m,l,k) = (m \wedge l) \vee (\neg m \wedge k)$, $K_t = 0x5A827999, 0 \leqslant t \leqslant 19$
2) $F_t(m,l,k) = m \oplus l \oplus k$, $K_t = 0x6ED9EBA1, 20 \leqslant t \leqslant 39$
3) $F_t(m,l,k) = (m \wedge l) \vee (m \wedge k) \vee (l \wedge k)$, $K_t = 0x8F1BBCDC, 40 \leqslant t \leqslant 59$
4) $F_t(m,l,k) = m \oplus l \oplus k$, $K_t = 0xCA62C1D6, 60 \leqslant t \leqslant 79$

Алгоритм SHA-1

У SHA-1 есть свое внутреннее 160-битное состояние, которое задается 32-битными словами H_0, H_1, H_2, H_3, H_4. Оно инициализируется следующим образом:

1) $H_0 = 0x67452301$
2) $H_1 = 0xEFCDAB89$
3) $H_2 = 0x98BADCFE$
4) $H_3 = 0x10325476$
5) $H_4 = 0xC3D2E1F0$

Входное сообщение M дополняется единицей, нулями и длиной сообщения в битах (64 бита). Нулей добавляется такое количество, которое сделает длину сообщения кратной 512. Далее алгоритм по очереди обрабатывает 512-битные блоки.

Обработка 512-битного блока проходит в несколько этапов:

1) Блок состоит из 16 32-битных слов W_t. Блок расширяется до 80 32-битных слов по следующей формуле: $W_t = (W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16}) << 1$.
2) $A = H_0, B = H_1, C = H_2, D = H_3, E = H_4$
3) Далее идет цикл по t: $0 \leqslant t \leqslant 79$
   - $TEMP = (A << 5) + F_t(B,C,D) + E + W_t + K_t$;
   - $E = D; D = C; C = (B << 30); B = A; A = TEMP$
   - $H_0 = H_0 + A, H_1 = H_1 + B, H_2 = H_2 + C, H_3 = H_3 + D, H_4 = H_4 + E$
4) $H_0 += A, H_1 += B, H_2 += C, H_3 += D, H_4 += E$

Операция $(A << n)$ означает циклический сдвиг A влево на n битов.

Конечное 160-битное состояние является выходом SHA-1.

6. SHA-2 (только SHA-256)

Семейство хеш-функции SHA-2 было разработано АНБ США и опубликовано NIST в 2002 году в стандарте FIPS PUB 180-2 [5].

Константы

В SHA-256 используются следующие константы:

```
k[0..63] :=
    0x428A2F98, 0x71374491, 0xB5C0FBCF, 0xE9B5DBA5, 0x3956C25B, 0x59F111F1, 0x923F82A4, 0xAB1C5ED5,
    0xD807AA98, 0x12835B01, 0x243185BE, 0x550C7DC3, 0x72BE5D74, 0x80DEB1FE, 0x9BDC06A7, 0xC19BF174,
    0xE49B69C1, 0xEFBE4786, 0x0FC19DC6, 0x240CA1CC, 0x2DE92C6F, 0x4A7484AA, 0x5CB0A9DC, 0x76F988DA,
    0x983E5152, 0xA831C66D, 0xB00327C8, 0xBF597FC7, 0xC6E00BF3, 0xD5A79147, 0x06CA6351, 0x14292967,
    0x27B70A85, 0x2E1B2138, 0x4D2C6DFC, 0x53380D13, 0x650A7354, 0x766A0ABB, 0x81C2C92E, 0x92722C85,
    0xA2BFE8A1, 0xA81A664B, 0xC24B8B70, 0xC76C51A3, 0xD192E819, 0xD6990624, 0xF40E3585, 0x106AA070,
    0x19A4C116, 0x1E376C08, 0x2748774C, 0x34B0BCB5, 0x391C0CB3, 0x4ED8AA4A, 0x5B9CCA4F, 0x682E6FF3,
    0x748F82EE, 0x78A5636F, 0x84C87814, 0x8CC70208, 0x90BEFFFA, 0xA4506CEB, 0xBEF9A3F7, 0xC67178F2
```

Рис. 4. Константы SHA-256

Алгоритм SHA-256

У SHA-256 есть внутреннее 256-битное состояние, которое задается 32-битными словами $H_0, H_1, H_2, H_3, H_4, H_5, H_6, H_7$. Оно инициализируется следующим образом:

1) $H_0 = 0x6A09E667$
2) $H_1 = 0xBB67AE85$
3) $H_2 = 0x3C6EF372$
4) $H_3 = 0xA54FF53A$
5) $H_4 = 0x510E527F$
6) $H_5 = 0x9B05688C$
7) $H_6 = 0x1F83D9AB$
8) $H_7 = 0x5BE0CD19$

Входное сообщение M дополняется единицей, нулями и длиной сообщения в битах (64 бита). Нулей добавляется такое количество, которое сделает длину сообщения кратной 512. Далее алгоритм по очереди обрабатывает 512-битные блоки.

Обработка 512-битного блока проходит следующим образом:

1) 512-битный блок разбивается на 16 32 битных слов: $w[0..15]$
2) Генерация дополнительных 48 32-битных блоков:

   for i := 16 to 63:

   so := (w[i-15] rotr 7) oplus ($w[i-15]$ rotr 18) oplus ($w[i-15]$ shr 3)

   s1 := (w[i-2] rotr 17) oplus ($w[i-2]$ rotr 19) oplus ($w[i-2]$ shr 10)

   $w[i] := w[i-16] + s0 + w[i-7] +$ s1
3) Инициализация временных переменных: $a := H_0, b := H_1, c := H_2, d := H_3, e := H_4, f := H_5, g := H_6, h := H_7$
4) Основной цикл алгоритма.

   for i := 0 to 63:

   — $\Sigma 0$:= (a rotr 2) oplus (a rotr 13) oplus (a rotr 22)

   — Ma := (a and b) oplus (a and c) oplus (b and c)

   — $t2 := \Sigma\theta + Ma$

- $\Sigma 1$:= (e rotr 6) oplus (e rotr 11) oplus (e rotr 25)
- Ch := (e and f) oplus ((not e) and g)
- $t1 := h + \Sigma 1 + Ch + k[i] + w[i]$
- $h := g$
- $g := f$
- $f := e$
- $e := d + t1$
- $d := c$
- $c := b$
- $b := a$
- $a := t1 + t2$

5) Суммирования состояния с очередным результатом: H_0 += a, H_1 += b, H_2 += c, H_3 += d, H_4 += e, H_5 += f, H_6 += g, H_7 += h

Примечание:

- shr - логический сдвиг вправо.
- rotr — циклический сдвиг вправо.

Конечное 256-битное состояние является выходом SHA-256.

7. SHA-3

SHA-3 (Keccak) - алгоритм хеширования переменной разрядности, который построен с помощью архитектуры "губка". Этот алгоритм впервые был опубликован в 2008 году, в 2012 году он стал победителем конкурса от NIST и в 2015 - новым стандартом, оформленным в FIPS 202.

Так как SHA-3 основан на "губке то для описания алгоритма достаточно описать параметры "губки функцию паддинга и функцию перестановки f.

Параметры "губки"

Для SHA-3 длина внутреннего состояния равна 1600 бит. Количество раундов функции f на один блок сообщения составляет 24. Длина выходного хеша может принимать значения :224, 256, 384, 512. r и c принимают соответствующие значения:

Type	Output length	Rate, r	Capacity, c
SHA3-224	224	1152	448
SHA3-256	256	1088	512
SHA3-384	384	832	768
SHA3-512	512	576	1024

Рис. 5. Таблица значений r, c для SHA-3

Функция паддинга

Для алгоритма необходимо, чтобы в него подавались блоки сообщения размера r, поэтому каждое сообщение M необходимо дополнить битами справа, чтобы оно было кратно r.

- Если |M| mod r = 0, то pad = "$10^{r-2}1$".

— Если |M| mod r = r-1, то pad = "$10^{r-1}1$".

— Иначе pad = "$10^{(r-2)-p}1$ где p = |M| mod r.

Необходимые функции для SHA-3

Состояние S отображается в трехмерный тензор A размера 5*5*64 по формуле: $A[i][j][k] = S[(5i + j) * 64 + k]$, где $0 <= i, j <= 4$ и $0 <= k <= 63$. После отработки функции f новый тензор A' переводится в S по той же формуле.

Рассмотрим вспомогательную функцию $rc(t)$ и функции θ, ρ, π, χ, ι:

Функция $rc(t)$

На вход она получает целое число t.

1) Если t mod 255 = 0, то return 1.
2) Обозначим R = [10000000].
3) for i = 1 to (t mod 255):
   а) R = 0 || R
   б) R[0] = R[0] ⊕ R[8]
   в) R[4] = R[4] ⊕ R[8]
   г) R[5] = R[0] ⊕ R[8]
   д) R[6] = R[0] ⊕ R[8]
   е) R = R[:8] (обрезаем крайний элемент)
4) Возвращаем R[0].

Далее везде, где не указано специально итерация по индексам будет проходить в этом диапазоне: $0 <= i, j <= 4$ и $0 <= k <= 63$

Функция $\theta(A)$

Для всех (i, k):

$C(i,k) = A[i,0,k] \oplus A[i,1,k] \oplus A[i,2,k] \oplus A[i,3,k] \oplus A[i,4,k]$

$D(i,k) = C[(i-1) \bmod 5, k] \oplus C[(i+1) \bmod 5, (k-1) \bmod w]$

Для всех (i, j, k): $A[i,j,k] = A[i,j,k] + D[i,k]$

Функция $\rho(A)$

Пусть (i, j) = (1, 0). Для t от 0 до 23:

1) Для всех k: $A[i,j,k] = A[i,j,(k-(t+1)(t+2)/2) mod 64]$
2) (i, j) = (j, (2i+3j) mod 5)

Функция $\pi(A)$

Для всех (i, j, k): $A[i,j,k] = A[(i+3j) mod 5, i, k]$

Функция $\chi(A)$

Для всех (i, j, k): $A[i,j,k] = A[i,j,k] \oplus ((A[(i+1) mod 5, j, k] \oplus 1) * A[(i+2) mod 5, j, k])$

Функция $\iota(A, i_r)$

1) Пусть RC нулевой массив длины 64.
2) Для i от 0 до 6: $RC[2^i - 1] = rc(i + 7 * i_r)$.
3) Для всех k: $A[0,0,k] = A[0,0,k] \oplus RC[k]$

Функция перестановки $f(S)$

1) Перевод состояния S в тензор A.
2) Для i_r от 0 до 23: $A = \iota(\chi(\pi(\rho(\theta(A)))), i_r)$.
3) Перевод тензора A в состояние S.

8. ГОСТ Р 34.11-94

ГОСТ Р 34.11-94 «Информационная технология. Криптографическая защита информации. Функция хэширования» - российский стандарт криптографической хеш-функции, который приняли в 1994 году. В 2013 году его отменили в РФ.

Эта хеш-функция основана на конструкции Меркла-Дамгарда с функцией сжатия f, которая принимает 2 блока по 256 бит и выдает блок длиной 256 бит. Рассмотрим функцию паддинга и функцию сжатия f.

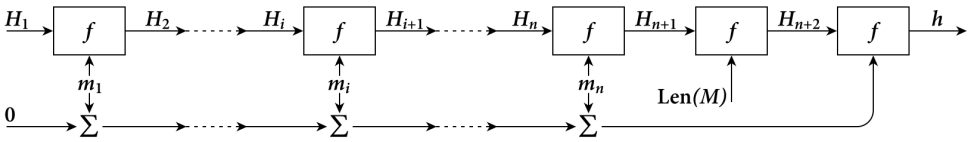

Рис. 6. Схема хеш-функции ГОСТ Р 34.11-94

Функция паддинга

Входное сообщение M делится на блоки по 256 бит, если крайний блок не кратен 256 бит, то он **слева** дополняется нулями так, чтобы длина блока была 256 бит. Затем добавляется блок в 256 бит, кодирующий изначальную длину сообщения. И еще добавляется блок, который является суммой по модулю 2 всех 256-битных блоков исходного сообщения.

Функция сжатия f

Функции сжатия $f(H_{i-1}, m) = H_i$ принимает 2 блока длиной 256 бит и выдает один блок длиной 256 бит. Эта функция состоит из трех частей: генерация ключей, шифрующее преобразование, перемешивание.

Генерация ключей

На этом этапе генерируется 4 ключа: K_1, K_2, K_3K_4.

Пусть есть 256-битный блок $Y = y_4||y_3||y_2||y_1$ или $Y = y_{32}||y_{31}||...||y_1$. Для генерации используется 2 функции, которые преобразуют 256-битный блок:

- $A(Y) = (y_1 \oplus y_2)||y_4||y_3||y_2$
- $P(Y) = y_{\varphi(32)}||y_{\varphi(31)}||...||y_{\varphi(31)}$, где $\varphi(i + 1 + 4(k - 1)) = 8i + k$ при $i = 0, ..., 3, k = 1, ..., 8$

Также используются константы:

- $C_2, C_4 = 0$
- $C_3 = 0xff00ffff000000ffff0000ff00ffff0000ff00ff00ff00ffff00ff00ff00ff00$

Алгоритм генерации ключей:

1) $U := H_{in}, \quad V := m, \quad W := U \oplus V, \quad K_1 = P(W)$
2) Для j от 2 до 4:
   $U := A(U) \oplus C_j, \quad V := A(A(V)), \quad W := U \oplus V, \quad K_j = P(W)$

Шифрующее преобразование

На этом этапе происходит шифрование $H_{in} = h_4||h_3||h_2||h_1$ с помощью сгенерированных ключей K_1, K_2, K_3, K_4. Шифрование происходит с помощью блочного шифра ГОСТ 28147—89 E [6]: $s_i = E(h_i, K_i)$, для i от 1 до 4. Выходом этого преобразования является: $S = s_4||s_3||s_2||s_1$

Перемешивание

На этом этапе 256-битный блок S и 256-битный блок сообщения m перемешиваются с помощью регистра сдвига с линейной обратной связью. РСЛОС определяется как $\psi(Y = y_{16}||y_{15}||...||y_1) = (y_1 \oplus y_2 \oplus y_3 \oplus y_4 \oplus y_{13} \oplus y_{16})||y_{16}||y_{15}||...||y_2$. В итоге перемешивающее образование имеет вид:

$H_{out} = \psi^{61}(H_{in} \oplus \psi(m \oplus \psi^{12}(S)))$

9. ГОСТ 34.11-2018 (Стрибог)

ГОСТ 34.11-2018 (Стрибог) «Информационная технология. Криптографическая защита информации. Функция хэширования» - межгосударственный криптографический стандарт хеш-функции, который приняли в 2018 году. В его основе лежит ГОСТ Р 34.11-2012, который в свою очередь пришел на смену устаревшему ГОСТ Р 34.11-94. Он может выдавать хеш размером 256 или 512 бит, рассмотрим 512-битный вариант.

Эта хеш-функция основана на конструкции Меркла-Дамгарда с функцией сжатия g, которая принимает 3 блока по 512 бит и выдает блок длиной 512 бит. Также $H_0 = IV = 0^{512}$. Рассмотрим функцию паддинга и функцию сжатия g.

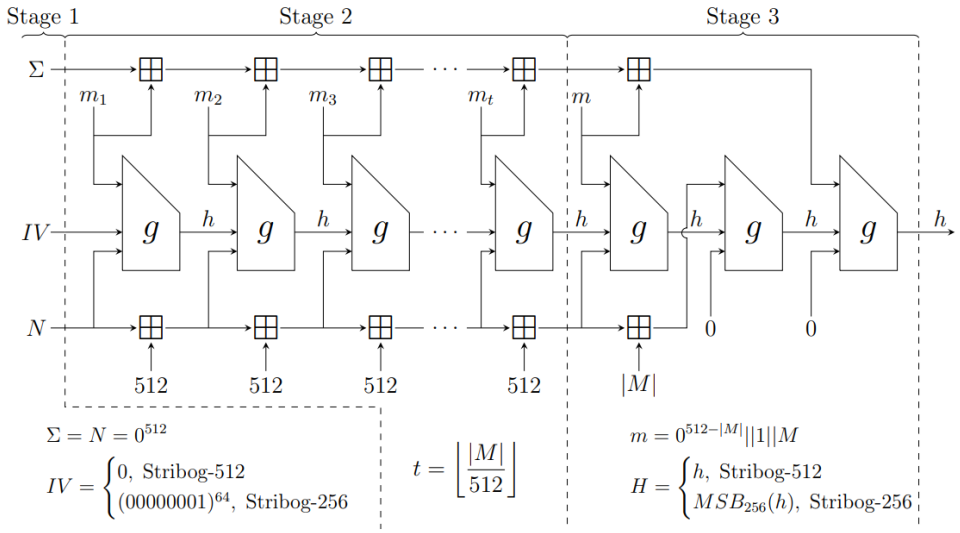

Рис. 7. Схема хеш-функции ГОСТ 34.11-2018 (Стрибог) [7]

Функция паддинга

Входное сообщение M делится на блоки по 512 бит, если крайний блок не кратен 512 бит, то он **слева** дополняется нулями и одной единицей (0^*1) так, чтобы длина блока была 512 бит. Затем добавляется блок в 512 бит, кодирующий изначальную длину сообщения. И еще добавляется блок, который является суммой по модулю 2 всех 512-битных блоков исходного сообщения.

Функция сжатия g

В ГОСТ 34.11-2018 функция сжатия g основана на конструкции Миагути - Пренеля. На вход она принимает N (сумма обработанных на текущий момент битов исходного сообщения в 512-битном представлении, для двух последних блоков $N = 0$) и 2 512-битных блока h и m. Рассмотрим $g_N(h, m)$:

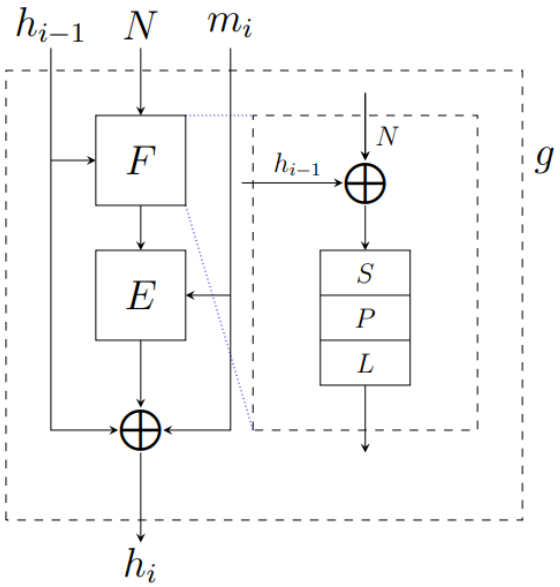

Рис. 8. Схема функции сжатия g в ГОСТ 34.11-2018 [7]

1) $g_N(h, m) = E(L \circ P \circ S(h \oplus N), m) \oplus h \oplus m$
2) $E(K, m) = X\left[K_{13}\right] \circ \prod_{i=1}^{12}\left(L \circ P \circ S \circ X\left[K_i\right]\right)(m))$
3) $K_i = L \circ P \circ S\left(K_{i-1} \oplus C_{i-1}\right), K_1 = K, i \in \{2, \ldots, 13\}$. Константы C_i можно найти в стандарте [8].
4) $X[K_i](A) = A \oplus K_i$
5) S (SubBytes) нелинейно преобразует каждые 8 бит в другие 8 бит с помощью перестановки (ее можно найти в стандарте [8]).
6) P (Permutation) это перестановка байт в 512 блоке (ее можно найти в стандарте [8]).
7) L (MixColumns) это линейное преобразование, при котором 512-битный блок представляется в виде матрицы размер 64x64 и умножается справа на специальную матрицу, операции проводятся по модулю 2 (ее можно найти в стандарте [8]).

Выходом алгоритма является крайнее значение h.

10. BLAKE-256

BLAKE - криптографическая хеш-функция [10], основанная на конструкции HAIFA и использующий идеи потокового шифра ChaCha. Эта функция оказалась финалистом конкурса NIST на SHA-3, уступив алгоритму Keccak в 1012 году. Существуют версии с выходом в 224, 256, 384 или 512 бит. Рассмотрим версию с 256-битным выходом BLAKE-256.

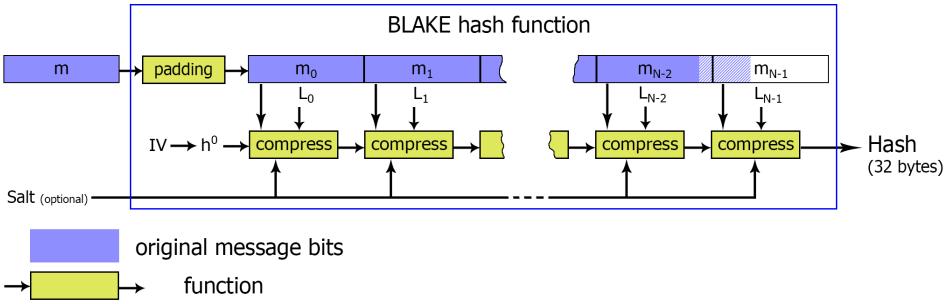

Рис. 9. Схема BLAKE-256 [9]

Константы и перестановки

В BLAKE-256 используются следующие константы:

2.1.1 Constants

BLAKE-256 starts hashing from the same initial value as SHA-256:

$IV_0 = 6A09E667$ $IV_1 = BB67AE85$
$IV_2 = 3C6EF372$ $IV_3 = A54FF53A$
$IV_4 = 510E527F$ $IV_5 = 9B05688C$
$IV_6 = 1F83D9AB$ $IV_7 = 5BE0CD19$

BLAKE-256 uses 16 constants[1]

$c_0 = 243F6A88$ $c_1 = 85A308D3$
$c_2 = 13198A2E$ $c_3 = 03707344$
$c_4 = A4093822$ $c_5 = 299F31D0$
$c_6 = 082EFA98$ $c_7 = EC4E6C89$
$c_8 = 452821E6$ $c_9 = 38D01377$
$c_{10} = BE5466CF$ $c_{11} = 34E90C6C$
$c_{12} = C0AC29B7$ $c_{13} = C97C50DD$
$c_{14} = 3F84D5B5$ $c_{15} = B5470917$

Рис. 10. Константы хеш-функции BLAKE-256 [10]

В BLAKE-256 используются следующие перестановки:

σ_0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
σ_1	14	10	4	8	9	15	13	6	1	12	0	2	11	7	5	3
σ_2	11	8	12	0	5	2	15	13	10	14	3	6	7	1	9	4
σ_3	7	9	3	1	13	12	11	14	2	6	5	10	4	0	15	8
σ_4	9	0	5	7	2	4	10	15	14	1	11	12	6	8	3	13
σ_5	2	12	6	10	0	11	8	3	4	13	7	5	15	14	1	9
σ_6	12	5	1	15	14	13	4	10	0	7	6	3	9	2	8	11
σ_7	13	11	7	14	12	1	3	9	5	0	15	4	8	6	2	10
σ_8	6	15	14	9	11	3	0	8	12	2	13	7	1	4	10	5
σ_9	10	2	8	4	7	6	1	5	15	11	9	14	3	12	13	0

Рис. 11. Перестановки S_16 хеш-функции BLAKE-256 [10]

Функция паддинга

BLAKE обрабатывает 512-битные блоки сообщения, поэтому необходимо добавить справа столько бит, чтобы длина сообщения стала кратна 512. Справа добавляется следующие биты: $pad = 10^*1L_{64}$, где L_{64} - это длина исходного сообщения в 64-битном представлении.

Функция сжатия *compress*

Функция сжатия *compress* принимает на вход 4 параметра:

— 256-битное состояние $h = h_0||...||h_7$

— 512-битный блок сообщения $m = m_0||...||m_{15}$

— 128-битный блок соли $s = s_0||...||s_3$

— 64-битный блок счетчика $t = t_0||t_1$

Этап инициализации. Начинается инициализация внутренних 32-битных переменных $v_0, \ldots, v_{15}$:

$$\begin{pmatrix} v_0 & v_1 & v_2 & v_3 \\ v_4 & v_5 & v_6 & v_7 \\ v_8 & v_9 & v_{10} & v_{11} \\ v_{12} & v_{13} & v_{14} & v_{15} \end{pmatrix} \leftarrow \begin{pmatrix} h_0 & h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 & h_7 \\ s_0 \oplus c_0 & s_1 \oplus c_1 & s_2 \oplus c_2 & s_3 \oplus c_3 \\ t_0 \oplus c_4 & t_0 \oplus c_5 & t_1 \oplus c_6 & t_1 \oplus c_7 \end{pmatrix}$$

Этап основной работы. Затем идет 14 раундов работы алгоритма над внутренними переменными. Один раунд выглядит так:

$$G_0\ (v_0, v_4, v_8, v_{12})\ G_1\ (v_1, v_5, v_9, v_{13})\ G_2\ (v_2, v_6, v_{10}, v_{14})\ G_3\ (v_3, v_7, v_{11}, v_{15})$$
$$G_4\ (v_0, v_5, v_{10}, v_{15})\ G_5\ (v_1, v_6, v_{11}, v_{12})\ G_6\ (v_2, v_7, v_8, v_{13})\ G_7\ (v_3, v_4, v_9, v_{14})$$

$G_i(a, b, c, d)$ работает так:

$$j \leftarrow \sigma_{r\%10}[2 \times i]$$
$$k \leftarrow \sigma_{r\%10}[2 \times i + 1]$$
$$a \leftarrow a + b + (m_j \oplus c_k)$$
$$d \leftarrow (d \oplus a) >>> 16$$
$$c \leftarrow c + d$$
$$b \leftarrow (b \oplus c) >>> 12$$
$$a \leftarrow a + b + (m_k \oplus c_j)$$
$$d \leftarrow (d \oplus a) >>> 8$$
$$c \leftarrow c + d$$
$$b \leftarrow (b \oplus c) >>> 7$$

Финальный этап. В конце подсчитывается выходное значение функции сжатия из внутренних переменных, соли и предыдущего выходного значения:

$$h'_0 \leftarrow h_0 \oplus s_0 \oplus v_0 \oplus v_8$$
$$h'_1 \leftarrow h_1 \oplus s_1 \oplus v_1 \oplus v_9$$
$$h'_2 \leftarrow h_2 \oplus s_2 \oplus v_2 \oplus v_{10}$$
$$h'_3 \leftarrow h_3 \oplus s_3 \oplus v_3 \oplus v_{11}$$
$$h'_4 \leftarrow h_4 \oplus s_0 \oplus v_4 \oplus v_{12}$$
$$h'_5 \leftarrow h_5 \oplus s_1 \oplus v_5 \oplus v_{13}$$
$$h'_6 \leftarrow h_6 \oplus s_2 \oplus v_6 \oplus v_{14}$$
$$h'_7 \leftarrow h_7 \oplus s_3 \oplus v_7 \oplus v_{15}$$

Таким образом, на выход идет 256-битный блок $h'_0||...||h'_7$.

11. RIPEMD-256

RIPEMD-256 - криптографическая хеш-функция, по факту состоящая из двух копий RIPEMD-128 и соответственно имеющая такой же уровень безопасности как RIPEMD-128. RIPEMD-128 была разработана в 1996 году.

RIPEMD-128 - криптографическая хеш-функция, построенная на конструкции Меркла-Дамгарда и имеющая 128-битный выход.

Функция паддинга

RIPEMD-128 работает с 512-битными блоками сообщения, поэтому нужно сделать длину сообщения кратной 512. Справа к сообщению добавляется единица, нули и длина сообщения, закодированная в 64-битном блоке. Нулей добавляется столько, чтобы длина сообщения стала кратна 512.

Необходимые константы и функции

Обозначим необходимые константы и функции для работы RIPEMD-128:

Table 3. Word permutations for the message expansion in RIPEMD-128

round j	$\pi_j^l(k)$																$\pi_j^r(k)$															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	5	14	7	0	9	2	11	4	13	6	15	8	1	10	3	12
1	7	4	13	1	10	6	15	3	12	0	9	5	2	14	11	8	6	11	3	7	0	13	5	10	14	15	8	12	4	9	1	2
2	3	10	14	4	9	15	8	1	2	7	0	6	13	11	5	12	15	5	1	3	7	14	6	9	11	8	12	2	10	0	4	13
3	1	9	11	10	0	8	12	4	13	3	7	15	14	5	6	2	8	6	4	1	3	11	15	0	5	12	2	13	9	7	10	14

Table 4. Boolean functions and round constants in RIPEMD-128, with $\mathrm{XOR}(x,y,z) := x \oplus y \oplus z$, $\mathrm{IF}(x,y,z) := x \wedge y \oplus \bar{x} \wedge z$ and $\mathrm{ONX}(x,y,z) := (x \vee \bar{y}) \oplus z$

round j	$\Phi_j^l(x,y,z)$	$\Phi_j^r(x,y,z)$	K_j^l	K_j^r
0	$\mathtt{XOR}(x,y,z)$	$\mathtt{IF}(z,x,y)$	0x00000000	0x50a28be6
1	$\mathtt{IF}(x,y,z)$	$\mathtt{ONX}(x,y,z)$	0x5a827999	0x5c4dd124
2	$\mathtt{ONX}(x,y,z)$	$\mathtt{IF}(x,y,z)$	0x6ed9eba1	0x6d703ef3
3	$\mathtt{IF}(z,x,y)$	$\mathtt{XOR}(x,y,z)$	0x8f1bbcdc	0x00000000

Table 5. Rotation constants in RIPEMD-128

round j	$s^l_{16\cdot j+k}$																$s^r_{16\cdot j+k}$															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	11	14	15	12	5	8	7	9	11	13	14	15	6	7	9	8	8	9	9	11	13	15	15	5	7	7	8	11	14	14	12	6
1	7	6	8	13	11	9	7	15	7	12	15	9	11	7	13	12	9	13	15	7	12	8	9	11	7	7	12	7	6	15	13	11
2	11	13	6	7	14	9	13	15	14	8	13	6	5	12	7	5	9	7	15	11	8	6	6	14	12	13	5	14	13	13	7	5
3	11	12	14	15	14	15	9	8	9	14	5	6	8	6	5	12	15	5	8	11	14	14	6	14	6	9	12	9	12	5	15	8

Рис. 12. Таблица функции и констант, необходимых для RIPEMD-128

Функция сжатия f

Функция сжатия $f(h,m)$ принимает на вход 128-битный блок $h = h_0||h_1||h_2||h_3$ и 512-битный блок сообщения m.

Этап инициализации. RIPEMD-128 имеет внутреннее 256-битное состояние, состоящее из двух так называемых ветвей: левой ветви X и правой ветви Y. Они инициализируются следующим образом:

$X_{-3} = h_0 \quad X_{-2} = h_1 \quad X_{-1} = h_2 \quad X_0 = h_3$

$Y_{-3} = h_0 \quad Y_{-2} = h_1 \quad Y_{-1} = h_2 \quad Y_0 = h_3$

Если это самая первая итерация функции сжатия, то ветви инициализируются так:

$X_{-3} = Y_{-3} =$ 0x67452301 $\quad X_{-2} = Y_{-2} =$ 0xefcdab89 $\quad X_{-1} = Y_{-1} =$ 0x98badcfe $\quad X_0 = Y_0 = 0 \times 10325476$

Этап расширения блока сообщения. Входной блок сообщения $m = m_0||...||m_15$ представляется в виде 16 32-битных слов. Заводится 2 массива длиной 64 элемента: W^l и W^r. Они заполняются следующим образом. Для всех (j, k): 0 <= j <= 3, 0 <= k <= 15:

- $W^l_{j\cdot 16+k} = M_{\pi^l_j(k)}$
- $W^r_{j\cdot 16+k} = M_{\pi^r_j(k)}$

Основной этап. Далее идет 64 шага, разделенных на 4 раунда по 16 шагов. 1 шаг выглядит так:

$$X_{i+1} = \left(X_{i-3} \boxplus \Phi^l_j\left(X_i, X_{i-1}, X_{i-2}\right) \boxplus W^l_i \boxplus K^l_j\right)^{\lll s^l_i}$$

$$Y_{i+1} = \left(Y_{i-3} \boxplus \Phi^r_j\left(Y_i, Y_{i-1}, Y_{i-2}\right) \boxplus W^r_i \boxplus K^r_j\right)^{\lll s^r_i}$$

Где i - номер шага ($o <= i <= 63$), а j - номер раунда ($o <= j <= 3$), $\ll k$ - циклический сдвиг влево на k.
Финальный этап. Выход функции сжатия $h^{'} = h_0^{'}|| \ldots ||h_3^{'}$ считается по следующей формуле:
$h_0' = X_{63} \boxplus Y_{62} \boxplus h_1 \quad h_1' = X_{62} \boxplus Y_{61} \boxplus h_2$
$h_2' = X_{61} \boxplus Y_{64} \boxplus h_3 \quad h_3' = X_{64} \boxplus Y_{63} \boxplus h_0$

12. Whirlpool

Whirlpool — криптографическая хеш-функция, первая версия которой появилась в 2000 году. Рассмотрим последнюю версию [11], которая вошла в стандарт ISO/IEC 10118-3:2004.

Whirlpool основана на конструкции Меркла-Дамгарда, функция сжатия основана на конструкции Миагути - Пренеля. Функция сжатия принимает 2 блока по 512-бит и выдает один 512-битный блок.

Функция паддинга

Необходимо, чтобы длина M была кратна 512. Для этого перед вычислением хеша сообщение обрабатывается следующим образом:

— К концу сообщения справа добавить единицу.

— Затем добавить такое количество нулей справа, чтобы итоговая длина сообщения была нечетное количество раз кратна 256.

— Добавить справа 256-битное представление длины исходного сообщения.

Алгоритм Whirlpool

После выравнивания сообщения M оно разбивается на t блоков по 512-бит m_i, которые по очереди обрабатываются по следующему алгоритму:

$\eta_i = \mu\left(m_i\right)$
$H_0 = \mu(IV)$
$H_i = W\left[H_{i-1}\right]\left(\eta_i\right) \oplus H_{i-1} \oplus \eta_i, 1 \leqslant i \leqslant t$
Здесь $IV = 0^{512}$. Выходом хеш-функции является 512-битный блок H_t.
Рассмотрим функции μ и блочный шифр W:

Функция μ

Функция $\mu(a)$ на вход принимает 512-битный вектор a и возвращает матрицу размера 8*8 внутри которой расположены байты. Преобразование происходит по следующей формуле:
$\mu(a) = b \Leftrightarrow b_{ij} = a_{8i+j}, 0 \leqslant i, j \leqslant 7$
Индексация в a происходит побайтово.

Вычисление ключей

Из ключа K, который является матрицей 8*8 вычисляются ключи по следующей формуле:
$K^0 = K$
$K^r = \rho\left[c^r\right]\left(K^{r-1}\right), r > 0$

Функция γ

Эта функция нелинейно преобразует каждый байт входной матрицы с помощью специального S-box'а, который описан в [11]:
$\gamma(a) = b \Leftrightarrow b_{ij} = S[a_{ij}], 0 \leqslant i, j \leqslant 7$

Функция π

Функция циклической перестановки π, которая задается следующим образом:

$\pi(a) = b \Leftrightarrow b_{ij} = a_{(i-j) \mod 8, j}, 0 \leqslant i, j \leqslant 7$

Функция θ

Функция линейной диффузии θ, которая входную матрицу умножает справа на специальную матрицу (все операции проходят по модулю 256):

$\theta(a) = b \Leftrightarrow b = a * C$

Матрица C выглядит так:

$$C = \begin{bmatrix} 01_x & 01_x & 04_x & 01_x & 08_x & 05_x & 02_x & 09_x \\ 09_x & 01_x & 01_x & 04_x & 01_x & 08_x & 05_x & 02_x \\ 02_x & 09_x & 01_x & 01_x & 04_x & 01_x & 08_x & 05_x \\ 05_x & 02_x & 09_x & 01_x & 01_x & 04_x & 01_x & 08_x \\ 08_x & 05_x & 02_x & 09_x & 01_x & 01_x & 04_x & 01_x \\ 01_x & 08_x & 05_x & 02_x & 09_x & 01_x & 01_x & 04_x \\ 04_x & 01_x & 08_x & 05_x & 02_x & 09_x & 01_x & 01_x \\ 01_x & 04_x & 01_x & 08_x & 05_x & 02_x & 09_x & 01_x \end{bmatrix}$$

Рис. 13. Матрица C хеш-фукнции Whirlpool [11]

Функция σ

Функция σ подмешивает ключ к входному аргументу:

$\sigma[k](a) = b \Leftrightarrow b_{ij} = a_{ij} \oplus k_{ij}, 0 \leqslant i, j \leqslant 7$

Раундовые константы c^r

Раундовые константы c^r задаются следующим образом:

$c^r_{0j} \equiv S[8(r-1)+j], \quad 0 \leqslant j \leqslant 7$

$c^r_{ij} \equiv 0, \quad 1 \leqslant i \leqslant 7, 0 \leqslant j \leqslant 7$

Функция $\rho[K]$

Функция $\rho[K]$ задается следующим образом:

$\rho[k] \equiv \sigma[k] \circ \theta \circ \pi \circ \gamma$

Блочный шифр $W[]()$

Блочный шифр $W[K](\eta)$ задается как параметризованная функция от K, которая преобразует входную матрицу размера 8*8 в выходную матрицу размера 8*8. Задается она следующим образом:

$W[K] = (\bigcirc_1^{r=R} \rho[K^r]) \circ \sigma\left[K^0\right]$ (композиция из 11 функций)

Ключи K_i вычисляются из K. Дефолтное количество раундов $R = 10$.

12.1. Криптоанализ. Rebound attack

В течении 8 лет с момента ее появления на эту хеш-функцию не было представлено ни одной атаки. И вот в 2009 году была опубликована первая атака нового типа - Rebound attack [13].

Rebound attack

Переобозначим функции в хеш-функции:

— σ = AK

— θ = SB

— π = SC

— γ = MR

— Соответственно раунд Whirlpool представляется как: $r := AK \circ MR \circ SC \circ SB$.

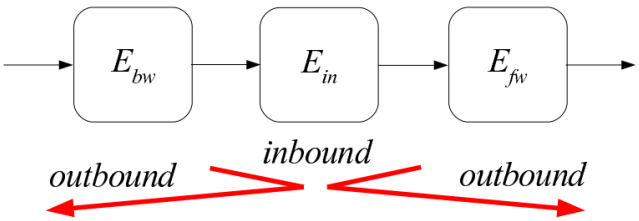

Рис. 14. Общая схема Rebound attack [13]

Атака делится на два концептуальных этапа:

1) **Inbound phase**: Фаза "встречи посередине"E_{in}, в котором есть достаточно пространство для подбора переменных для нахождения "встречи"(match).

2) **Outbound phase**: В этой фазе используются усеченные дифференциалы, который как раз находятся в первой фазе, и распространяются вперед и назад в сторону E_{fw} и E_{bw} Соответственно . Если усеченные дифференциалы имеют низкую вероятность, тогда необходимо увеличить количество итерации в первой фазе, чтобы их найти. Таким образом находится вероятный дифференциал между входом и выходом алгоритма, с помощью которого строится коллизия, полусвободная коллизия или полусвободная почти коллизия.

Авторы описали 3 атаки: на 4.5, 5.5 и 7.5 раундов. Рассмотрим атаку на 4.5 раундов.

Основная идея атаки - это найти последовательность дифференциалов, которая проходит через несколько активных S-box'ов: (1, 8, 64, 8, 1) S-box'ов Соответственно. Начало последовательности совпадает со входом в хеш-функцию и конец с концом 4.5 раунда. E_{bw}, E_{in}, E_{fw} определяются следующим образом:

— $E_{bw} = SC \circ SB \circ AK \circ MR \circ SC \circ SB$

— $E_{in} = MR \circ SC \circ SB \circ AK \circ MR$

— $E_{fw} = AK \circ MR \circ SC \circ SB \circ AK$

12.1.1. Этап предварительных вычислений

Предварительно вычисляются все вероятные дифференциалы для S-box Whirlpool. Вычисляются все 256*256 разностей (x,y) и соответствующие им дифференциалы. Примерно половина из них вероятные, поэтому вероятность, что разность даст вероятный дифференциал берется за $1/2$. Распределение дифференциалов для S-box можно найти в работе [11].

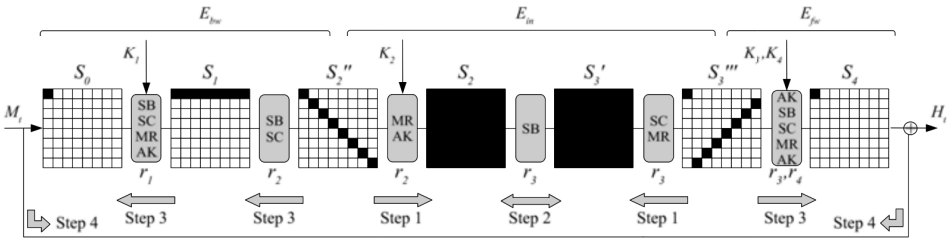

Fig. 5. A schematic view of the attack on 4 rounds of Whirlpool with round key inputs and feed-forward. Black state bytes are active.

Рис. 15. Графическое представление Rebound attack [11]

12.1.2. Inbound phase. 1 этап

Выбирается случайная разность с 8 активными байтами состояния (то есть 8 ненулевых байтов, остальные нулевые) S_2'' перед слоем MR раунда r_2. Все активные байты должны находиться на диагонали состояния S_2'' (см. рисунок выше). Затем разности распространяются вперёд до полностью активного состояния на входе следующего слоя SB (состояние S_2) с вероятностью 1. Далее мы выбираем другую случайную разность с 8 активными байтами в состоянии S_3''' после функции MR раунда r_3 и распространяем её назад. Снова диагональная форма гарантирует, что мы получаем полностью активное состояние на выходе SB раунда r_3.

12.1.3. Inbound phase. 2 этап

Это шаг, который реализует "встречу посередине". Мы ищем совпадающую входную и выходную разность (вероятный дифференциал) для слоя SB раунда r_3, используя заранее вычисленную таблицу дифференциалов S-бокса. Поскольку вероятность нахождения совпадения для каждого байта составляет 1/2, вероятность нахождения дифференциала для всего активного слоя SubBytes составляет примерно 2^{-64}. Таким образом, после повторения шага 1 атаки примерно 2^{64} раз мы ожидаем найти дифференциал SubBytes для всего состояния (все 64 байта одновременно имеют дифференциал). Мы получаем как минимум два значения состояния (слева и справа от SB) для каждого совпадения S-бокса, у нас будет около 2^{64} начальных точек для выходного этапа. Эти 2^{64} начальных точек можно сконструировать с общей сложностью около 2^{64}.

12.1.4. Outbound phase. 3 этап

В outbound phase мы продолжаем расширять дифференциальную последовательность назад и вперёд. Продвигая вероятный дифференциал слева через следующий слой SubBytes, мы получаем усечённый дифференциал с 8 активными байтами в каждом направлении. Затем усечённые дифференциалы должны распространяться от 8 до 1 активного байта через преобразование MR, как в обратном, так и в прямом направлении.

В [11] показано что, для преобразования 8 активных байтов в 1 активный байт при прохождении через MR в среднем необходимо перебрать 2^{56} начальных точек (2^{7*8}, где 2^8 - обратная вероятность появления нулевого значения и 7 их количество). Так как, нужно пройти в 2 направления, то необходимо перебрать $2^{56*2} = 2^{112}$ начальных точек. У нас есть необходимое количество попыток, так как в indound фазе мы можем перебирать до 2^{128} точек.

12.1.5. Outbound phase. 4 этап

Осталось 2 активных байтов по краям и необходимо, чтобы они были одинаковыми чтобы разность на входе и на выходе стали одинаковыми. Вероятность появления одинаковых байтов равна $2 * -8$. Поэтому необходимо 2^8 раз повторить последний этап. Итого сложность атаки получилась $2^{112+8} = 2^{120}$ операции Whirlpool. Требуемые объем памяти - 2^{16}. Заметим, что подмешивание ключей влияния не оказывает. Также атака работает со стандартным IV. Авторы отмечают, что можно добавить еще полраунда SB, SC, так как они не ломают построенный дифференциал. По факту атака проводится над 4 раундами, но получается, что можно добавить еще пару функции и выйдет 4.5 раунда.

12.1.6. Rebound атака на 5.5 раундов

Rebound атака на 5.5 раундов выглядит так:

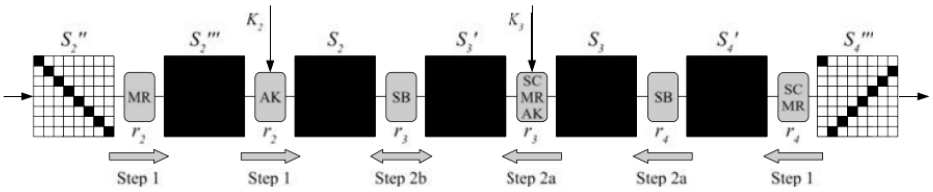

Fig. 6. In the attack on 5.5 rounds we first choose random values of the state S_4' to propagate backwards (Step 2a) and then, use the degrees of freedom from the key schedule to solve the difference propagation of the S-box in round r_3 (Step 2b).

Рис. 16. Rebound атака на 5.5 раундов [11]

Количество операции - 2^{120}, количество памяти - 2^{16}.

12.1.7. Rebound атака на 7.5 раундов

Rebound атака на 7.5 раундов выглядит так:

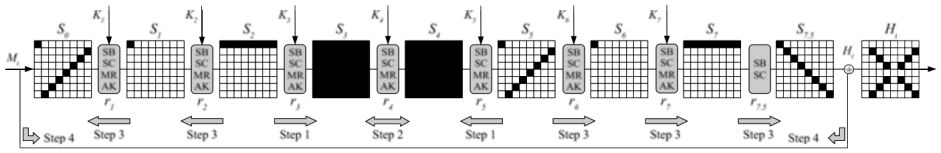

Fig. 7. In the attack on 7.5 rounds we extend the trail by one more round at the beginning and 1.5 rounds at the end to get a semi-free-start near-collision of Whirlpool

Рис. 17. Rebound атака на 7.5 раундов [11]

Количество операции - 2^{128}, количество памяти - 2^{16}.

12.1.8. Вывод

Таким образом, эти атаки не представляют практической угрозы нахождения коллизии, так как требуют много операции и работают лишь на определенном количестве раундов. Однако rebound атака имеет значительное теоретическое значение, так как открыло новый вектор исследования криптографических схем.

12.2. Заключение

Хеш-функция Whirlpool, основанная на конструкции Меркла-Дамгарда и использующая блочный шифр, на протяжении восьми лет оставалась устойчивой к атакам. Однако в 2009 году была предложена новая атака — Rebound attack, которая позволила находить коллизии на частичных раундах Whirlpool.

13. Fast Syndrome Based Hash (FSB)

FSB (Fast Syndrome-Based Hash Function) — это семейство криптографических хеш-функций [12], которое разработали в 2003 году.

FSB использует конструкцию Меркла-Дамгарда с классическим паддингом. Опишем функцию сжатия f.

Функция сжатия f

На вход эта функция принимает блок длиной s, который состоит из предыдущего значения h ($|h| = r$) и блока сообщения m ($|m| = s - r$). Внутри себя она использует булеву матрицу $\mathcal{H}$ размера $r * n$, которая сгенерирована

из равномерного распределения над всеми булевыми матрицами соответствующего размера. Должно соблюдаться следующее:

1) $n, r, w, s, log_2(n/w)$ должны быть натуральными числами
2) $s = w * log_2(n/w)$
3) $s > r$

Матрица $\mathcal{H}$ разбивается на w подматриц $\mathcal{H}_i$ размера $r * (n/w)$. Затем происходит основной алгоритм:

1) Входной блок s разбивается на w подряд идущих векторов, каждый из которых отображается в то неотрицательное число, двоичной записью которого этот вектор является. Таким образом s переходит в $(s_1, \ldots, s_w)$.
2) Далее из каждой подматрицы $\mathcal{H}_i$ выбирается s_i столбец. Затем эти столбцы складываются по модулю 2 и сохраняется в переменную h.
3) На выход подается h, длина h равна r.

14. Список литературы

[1] Миронкин В. О. Об одной теоретико-вероятностной модели Sponge-конструкции //Обозрение прикладной и промышленной математики. – 2018. – Т. 25. – №. 1. – С. 3-8.

[2] Rivest R. The MD5 message-digest algorithm. – 1992. – №. rfc1321.

[3] Standard S. H. Secure hash standard //FIPS PUB. – 1995. – С. 180-1.

[4] Eastlake 3rd D., Jones P. US secure hash algorithm 1 (SHA1). – 2001. – №. rfc3174.

[5] Secure hash standard //FIPS PUB 180-2. – 2002.

[6] ГОСТ 28147-89 на сайте ФГУП «Стандартинформ» // URL: https://www.gostinfo.ru/catalog/Details/?id=4149371

[7] Kazymyrov O., Kazymyrova V. Algebraic aspects of the russian hash standard GOST R 34.11-2012 //Cryptology ePrint Archive. – 2013.

[8] ГОСТ 34.11-2018 «Информационная технология. Криптографическая защита информации. Функция хэширования» // URL: http://protect.gost.ru/v.aspx?control=7id=232143

[9] Block-diagram of BLAKE hash function algorythm // URL: https://commons.wikimedia.org/wiki/File:BLAKE$_algorythm.png$

[10] Aumasson J. P. et al. Sha-3 proposal blake //Submission to NIST. – 2008. – Т. 92. – С. 194.

[11] Barreto P. et al. The Whirlpool hashing function //First open NESSIE Workshop, Leuven, Belgium. – 2000. – Т. 13. – С. 14.

[12] Augot D., Finiasz M., Sendrier N. A fast provably secure cryptographic hash function //Cryptology ePrint Archive. – 2003.

[13] Mendel F. et al. The rebound attack: Cryptanalysis of reduced Whirlpool and Grøstl //Fast Software Encryption: 16th International Workshop, FSE 2009 Leuven, Belgium, February 22-25, 2009 Revised Selected Papers. – Springer Berlin Heidelberg, 2009. – С. 260-276.