2 Забезпечення комп'ютерної безпеки в державних, банківських та інших інформаційних системах

УДК 681.3.06 : 519.248.681

БЛОЧНЫЙ СИММЕТРИЧНЫЙ КРИПТОАЛГОРИТМ SHACAL-2

Геннадий Гулак*, Иван Горбенко, Матвей Михайленко, Юрий Гитис
Харьковский Национальный Университет Радиоэлектроники
**ДСТСЗИ СБУ*

Анотація: **Наводиться опис 256-бітного блокового симетричного шифру SHACAL-2. Наводяться результати аналізу основних характеристик алгоритму.**
Summary: **The 256-bit symmetric block cipher SHACAL-2 is described. The results of the main characteristics are cited.**
Ключові слова: **Криптоалгоритм SHACAL-2, хэш-перетворення, тестування алгоритмів.**

Введение

Успешно завершился научно-исследовательский проект Nessie. Начиная с сентября 2000 года, криптографы из более чем 10 стран со всего мира представили 42 криптоалгоритма. Проект рассматривал алгоритмы зашифрования трех типов: блочные шифры, поточные шифры и шифры с открытым ключом зашифрования. Также принимали участие цифровые подписи (в комбинации с хэш-функциями), протоколы аутентификации, генераторы случайных чисел, схемы быстрой аутентификации пакетов данных, хэш-функции и алгоритмы цифровой подписи. В качестве основных критериев отбора претендентов названы безопасность, производительность, гибкость и требования рынка.

С тех пор исследователи внутри и вне проекта Nessie осуществляли атаки на эти алгоритмы, делая попытку найти слабости, которые ставили бы под угрозу их безопасность. Кроме того, была оценена эффективность этих алгоритмов. Как следствие этой оценки, выбор из 42 соперников был уменьшен до 24 кандидатов в сентябре 2001. Вторая стадия выбора, закончившаяся в марте 2003, уменьшила это число до 12. В классе блочных симметричных шифров были отобраны MISTY I, Camellia, AES (Advanced Encryption Standard) (USA FIPS 197) (Rijndael) и рассматриваемый в данной статье SHACAL-2.

I Общая характеристика криптоалгоритма

SHACAL-2 основан на хэш-стандарте SHA-2, который был введен NIST в 2000 году [1, 2]. Несмотря на сходство в имени с SHACAL-1, можно сказать, что SHACAL-2 – совершенно иной алгоритм. Это 256-битный блочный шифр с 512-битовым ключом, хотя он может быть сконфигурирован для использования 512-битовых блоков.

SHA-2 – новый проект, который имеет некоторое сходство с SHA-1, но есть важные различия в структуре. Можно отметить, что функция сжатия в SHA-2 является 64-шаговой одноцикловой, в отличии от четырехцикловой 80-шаговой (4 цикла по 20 шагов) в SHA-1. Таким образом, число шагов, входящих в функцию сжатия, меньше, чем в SHA-1 (64 по сравнению с 80). С другой стороны, две переменные обновляются на каждом шаге функции сжатия, а в SHA-1 только одна переменная обновляется в каждом шаге.

II SHACAL-2 в режиме шифрования

Функция сжатия SHA-2 может быть использована в шифровальном режиме, используя ключ как сообщение и открытый текст как начальную величину. Далее представлен алгоритм шифрования SHACAL-2.

2.1 Схема работы

Хеширование сообщения осуществляется следующим образом.

1. Дополнить сообщение так, чтобы его длина была кратной и соответствовала размеру функции сжатия, см. [4].

2. Инициализировать цепные переменные *A0, B0, C0, D0, E0, F0, G0, H0*, каждое 32-битовой длины, путем

A0 = IV1 = 6a09e667x,

B0 = IV2 = bb67ae85x,
C0 = IV3 = 3c6ef372x,
D0 = IV4 = a54ff53ax,
E0 = IV5 = 510e527fx,
F0 = IV6 = 9b05688cx,
G0 = IV7 = 1f83d9abx,
H0 = IV8 = 5be0cd19x.

3. Для каждого 512-битового блока сообщения:

(a) установить $A0 = A, B0 = B, C0 = C, D0 = D, E0 = E, F0 = F, G0 = G, H0 = H;$

(b) расширить 512 битов до 2048 битов (см. далее);

(c) сжать 2048 битов всего за 64 шага; каждый шаг модифицирует, в свою очередь, одну из рабочих переменных *A, B, C, D, E, F, G, H* (см. раздел по функции сжатия);

(d) установить $AA = A^{64} + A0, BB = B^{64} + B0, CC = C^{64} + C0, DD = D^{64} + D0, EE = E^{64} + E0, FF = F^{64} + F0, GG = G^{64} + G0, HH = H^{64} + H0.$

4. Вывод хэш-значения

$$[AA||BB||CC||DD||EE||FF||GG||HH].$$

2.2 Функция сжатия

Обозначим 512-битовые блоки сообщения $M = [W^0||W^1||...||W^{15}]$, где W^i — 32 -битовые слова.

Для SHACAL-2 определяется расширение 512 битов в 2048 битов, и затем ключ является конкатенацией 16 32-битовых слов: $M = [W^0||W^1||...||W^{15}]$.

В SHA-2 определяется расширение

$$W^i = \sigma_1(W^{i-2}) \oplus W^{i-7} \oplus \sigma_0(W^{i-15}) \oplus W^{i-16}, \; 16 \le i \le 63 \tag{1}$$

где σ_0 и σ_1 – определяется так:

$$\sigma_0(x) = S_7(x) \oplus S_{18}(x) \oplus R_3(x), \tag{2}$$

$$\sigma_1(x) = S_{17}(x) \oplus S_{19}(x) \oplus R_{10}(x), \tag{3}$$

где S_i - циклический сдвиг вправо на *i* бит, а R_i - сдвиг вправо на *i* бит. Ключи короче чем 512 битов могут быть дополнены нулями вплоть до 512 битов.

Помещаем 256-битовый открытый текст в восемь 32-битовых переменных *A, B, C, D, E, F, G, H.*

Для 64 шагов имеем

$$A^i = T^1 + T^2, \tag{4}$$

$$B^{i+1} = A^i, \tag{5}$$

$$C^{i+1} = B^i, \tag{6}$$

$$D^{i+1} = C^i, \tag{7}$$

$$E^{i+1} = D^i + T^i, \tag{8}$$

$$F^{i+1} = E^i, \tag{9}$$

$$G^{i+1} = F^i, \tag{10}$$

$$H^{i+1} = G^i, \tag{11}$$

$$T_1 = H + \Sigma_1(E) + Ch(E\ F\ G) + K^i + W^i, \tag{12}$$

$$T_2 = \Sigma_0(A) + Maj(A\ B\ C), \tag{13}$$

где W^i – 32-битовые шаговые ключи, K^i – константы, различные на каждом шаге, и

$$Ch(XYZ) = (X \,\&\, Y) \oplus (\overline{X} \,\&\, Z), \tag{14}$$

$$Maj(XYZ) = (X \,\&\, Y) \oplus (X \,\&\, Z) \oplus (Y \,\&\, Z), \tag{15}$$

$$\Sigma_0(X) = S_2(X) \oplus S_{13}(X) \oplus S_{22}(X), \tag{16}$$

$$\Sigma_1(X) = S_6(X) \oplus S_{11}(X) \oplus S_{25}(X), \tag{17}$$

где S_i - циклический сдвиг вправо на *i* бит.

Результат выполнения 64 шагов – A^{64}, B^{64}, C^{64}, D^{64}, E^{64}, F^{64}, G^{64}, H^{64}. Каждый из вышеуказанных 64 шагов является обратимым для переменных *A, B, C, D, E, F, G, H*. Таким образом, если вставить секретный ключ в сообщение и открытый текст как начальное значение, то можно получить обратимую функцию из функции сжатия путем игнорирования конечного сложения с начальными значениями.

III Схема работы SHA-2

Поскольку стойкость SHACAL-2 напрямую зависит от стойкости хэш-функции SHA-2, рассмотрим подробно устройство самой хэш-функции.

SHA-256 работает с блоками длиной меньше 2^{60} бит, хэш-функции SHA-384 и SHA-512 работают с длинами блоков, не превышающих 2^{128} бит. На выходе имеем последовательность длиной от 160 до 512 бит в зависимости от алгоритма. Вообще хэш-алгоритмы часто используются в других криптографических алгоритмах, таких как цифровая подпись, быстрая аутентификация пакетов, выработка случайной числовой последовательности.

Каждый из этих хэш-алгоритмов состоит из двух основных частей: предварительная обработка и подсчет хэш-суммы. Предварительная обработка в свою очередь состоит из формирования пакетов, анализа сформированных пакетов и инициализации хэш-переменных.

3.1 Используемые функции

3.1.1 Функции, используемые в SHA-256

SHA-256 использует шесть логических функций, каждая функция работает с 32-битными словами, содержащимися в переменных *x, y* и *z*. Результат каждой функции представляет собой новое 32-битное слово:

$$Ch(xyz) = (x \wedge y) \oplus (\bar{z} \wedge z), \tag{18}$$

$$Maj(xyz) = (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z), \tag{19}$$

$$\sum_0^{256}(x) = ROTR^2(x) \oplus ROTR^{13}(x) \oplus ROTR^{22}(x), \tag{20}$$

$$\sum_1^{256}(x) = ROTR^6(x) \oplus ROTR^{11}(x) \oplus ROTR^{25}(x), \tag{21}$$

$$\sigma_0^{256}(x) = ROTR^7(x) \oplus ROTR^{18}(x) \oplus SHR^3(x), \tag{22}$$

$$\sigma_1^{256}(x) = ROTR^{17}(x) \oplus ROTR^{19}(x) \oplus SHR^{10}(x), \tag{23}$$

где $ROTR^n(x)$ – циклический сдвиг *x* вправо на *n* бит, $SHR^n(x)$ – сдвиг *x* вправо на *n* бит.

3.1.2 Функции, используемые в SHA-384 и SHA-512

SHA-384 и SHA-512 используют шесть логических функций, каждая функция работает с 64-битными словами, содержащимися в переменных *x, y* и *z*. Результат каждой функции представляет собой новое 64-битное слово.

$$Ch(xyz) = (x \wedge y) \oplus (\bar{z} \wedge z), \tag{24}$$

$$Maj(xyz) = (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z), \tag{25}$$

$$\sum_0^{512}(x) = ROTR^{28}(x) \oplus ROTR^{34}(x) \oplus ROTR^{39}(x), \tag{26}$$

$$\sum_1^{512}(x) = ROTR^{14}(x) \oplus ROTR^{18}(x) \oplus ROTR^{41}(x), \tag{27}$$

$$\sigma_0^{512}(x) = ROTR^1(x) \oplus ROTR^8(x) \oplus SHR^7(x), \tag{28}$$

$$\sigma_1^{512}(x) = ROTR^{19}(x) \oplus ROTR^{61}(x) \oplus SHR^6(x), \tag{29}$$

3.2 Используемые константы

3.2.1 Константы, используемые в SHA-256

В SHA-256 используются 64 32-битных констант, $K_0^{(256)}, K_1^{(256)}, \ldots, K_{63}^{(256)}$. Константы представляют собой первые 32 бита от дробных частей кубических корней из 64 простых чисел. В 16-ричной системе счисления эти константы выглядят следующим образом:

```
428a2f98 71374491 b5c0fbcf e9b5dba5 3956c25b 59f111f1 923f82a4 ab1c5ed5
d807aa98 12835b01 243185be 550c7dc3 72be5d74 80deb1fe 9bdc06a7 c19bf174
e49b69c1 efbe4786 0fc19dc6 240ca1cc 2de92c6f 4a7484aa 5cb0a9dc 76f988da
983e5152 a831c66d b00327c8 bf597fc7 c6e00bf3 d5a79147 06ca6351 14292967
27b70a85 2e1b2138 4d2c6dfc 53380d13 650a7354 766a0abb 81c2c92e 92722c85
a2bfe8a1 a81a664b c24b8b70 c76c51a3 d192e819 d6990624 f40e3585 106aa070
19a4c116 1e376c08 2748774c 34b0bcb5 391c0cb3 4ed8aa4a 5b9cca4f 682e6ff3
748f82ee 78a5636f 84c87814 8cc70208 90befffa a4506ceb bef9a3f7 c67178f2
```

3.2.2. Константы, используемые в SHA-384 и SHA-512

В SHA-384 и SHA-512 используются 80 64-битных констант, $K_0^{(512)}, K_1^{(512)}, ..., K_{79}^{(512)}$. Константы представляют собой первые 64 бита от дробных частей кубических корней из 80 простых чисел. В 16-ричной системе счисления эти константы выглядят следующим образом:

```
428a2f98d728ae22 7137449123ef65cd b5c0fbcfec4d3b2f e9b5dba58189dbbc
3956c25bf348b538 59f111f1b605d019 923f82a4af194f9b ab1c5ed5da6d8118
d807aa98a3030242 12835b0145706fbe 243185be4ee4b28c 550c7dc3d5ffb4e2
72be5d74f27b896f 80deb1fe3b1696b1 9bdc06a725c71235 c19bf174cf692694
e49b69c19ef14ad2 efbe4786384f25e3 0fc19dc68b8cd5b5 240ca1cc77ac9c65
2de92c6f592b0275 4a7484aa6ea6e483 5cb0a9dcbd41fbd4 76f988da831153b5
983e5152ee66dfab a831c66d2db43210 b00327c898fb213f bf597fc7beef0ee4
c6e00bf33da88fc2 d5a79147930aa725 06ca6351e003826f 142929670a0e6e70
27b70a8546d22ffc 2e1b21385c26c926 4d2c6dfc5ac42aed 53380d139d95b3df
650a73548baf63de 766a0abb3c77b2a8 81c2c92e47edaee6 92722c851482353b
a2bfe8a14cf10364 a81a664bbc423001 c24b8b70d0f89791 c76c51a30654be30
d192e819d6ef5218 d69906245565a910 f40e35855771202a 106aa07032bbd1b8
19a4c116b8d2d0c8 1e376c085141ab53 2748774cdf8eeb99 34b0bcb5e19b48a8
391c0cb3c5c95a63 4ed8aa4ae3418acb 5b9cca4f7763e373 682e6ff3d6b2b8a3
748f82ee5defb2fc 78a5636f43172f60 84c87814a1f0ab72 8cc702081a6439ec
90befffa23631e28 a4506cebde82bde9 bef9a3f7b2c67915 c67178f2e372532b
ca273eceea26619c d186b8c721c0c207 eada7dd6cde0eb1e f57d4f7fee6ed178
06f067aa72176fba 0a637dc5a2c898a6 113f9804bef90dae 1b710b35131c471b
28db77f523047d84 32caab7b40c72493 3c9ebe0a15c9bebc 431d67c49c100d4c
4cc5d4becb3e42b6 597f299cfc657e2a 5fcb6fab3ad6faec 6c44198c4a475817
```

Теперь рассмотрим каждую из двух частей алгоритма.

3.3 Предварительная обработка

Перед началом подсчета хэш-суммы необходимо провести предварительную обработку информационных блоков сообщения, а именно: сформировать пакеты, разбить сформированные пакеты на части, инициализировать начальные значения $H^{(0)}$.

3.3.1 Формирование пакетов в SHA-256

Возьмем сообщение *M* длиной *l* бит. Запишем *M* в бинарном виде и добавим “1” к концу сообщения, далее следует последовательность нулей длиной *k*, где *k* вычисляется по формуле $l+1+k \equiv 448 (\mathrm{mod} 512)$. В конце добавляем блок данных, в котором записано значение *l* в бинарном виде и слева этот блок дополнен нулями до 64 бит. Например, сообщение *M* “abc”, записанное 8-битными символами в кодировке ASCII имеет длину $l = 8*3 = 24$ бит, *k* будет равно $k = 448 - (24+1) = 423$, а общая длина пакета 512 бит. Пример пакета отображен на рис. 1.

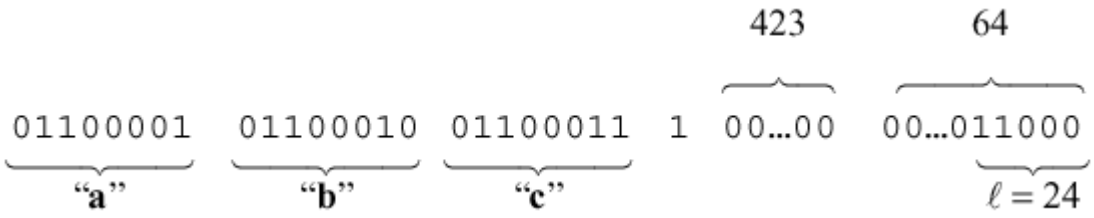


Рисунок 1 – Пример пакета SHA-256

3.3.2 Формирование пакетов в SHA-384 и SHA-512

Возьмем сообщение *M* длиной *l* бит. Запишем *M* в бинарном виде и добавим “1” к концу сообщения, далее следует последовательность нулей длиной *k*, где *k* вычисляется по формуле $l+1+k \equiv 896 (\mathrm{mod} 1024)$. В конце добавляем блок данных, в котором записано значение *l* в бинарном виде и слева этот блок дополнен нулями до 128 бит. Например, сообщение *M* “abc”, записанное 8-битными символами в кодировке ASCII имеет длину $l = 8*3 = 24$ бит, *k* будет равно $k = 896 - (24+1) = 871$, а общая длина пакета 1024 бит. Пример пакета отображен на рис. 2.

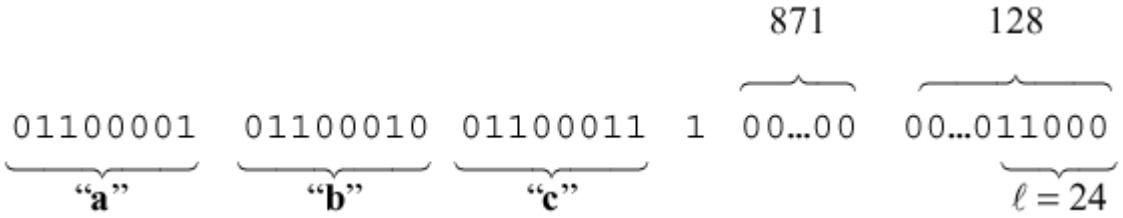


Рисунок 2 – Пример пакета SHA-384 и SHA-512

3.4 Анализ сформированных пакетов

В хэш-алгоритме SHA-256 все сообщение должно быть поделено на *N* пакетов по 512-бит Таким образом, в одном пакете может быть максимум 55 8-битных символов в кодировке ASCII. 512-битные блоки представляются как шестнадцать 32-битных слов. Обозначим первые 32 бита блока *i* как $M_0^{(i)}$, следующие 32 бита как $M_1^{(i)}$, и так далее до $M_{15}^{(i)}$.

В SHA-384 и SHA-512 все сообщение должно быть поделено на *N* пакетов по 1024-бит, таким образом в одном пакете может быт максимум 111 8-битных символов в кодировке ASCII. 1024-битные блоки представляются как шестнадцать 64-битных слов. Обозначим первые 64 бита блока *i* как $M_0^{(i)}$, следующие 64 бита как $M_1^{(i)}$, и так далее до $M_{15}^{(i)}$.

3.5 Инициализация хэш-переменных

Для SHA-256 инициализируем хэш-переменные $H^{(0)}$ значениями, состоящими из восьми 32-битных слов, которые в hex выглядят так:

$H_0^{(0)} = 6a09e667,$
$H_1^{(0)} = bb67ae85,$
$H_2^{(0)} = 3c6ef372,$
$H_3^{(0)} = a54ff53a,$
$H_4^{(0)} = 510e527f,$
$H_5^{(0)} = 9b05688c,$
$H_6^{(0)} = 1f83d9ab,$
$H_7^{(0)} = 5be0cd19.$

Эти значения представляют собой первые 32 бита дробных частей квадратных корней из восьми простых чисел.

Для SHA-384 инициализируем хэш-переменные $H^{(0)}$ значениями, состоящими из восьми 64-битных слов, которые в hex выглядят так:

$H_0^{(0)} = cbbb9d5dc1059ed8,$
$H_1^{(0)} = 629a292a367cd507,$
$H_2^{(0)} = 9159015a3070ddl7,$
$H_3^{(0)} = 152fecd8f70e5939,$
$H_4^{(0)} = 67332667ffc00b3l,$
$H_5^{(0)} = 8eb44a8768581511,$
$H_6^{(0)} = db0c2e0d64f98fa7,$
$H_7^{(0)} = 47b5481dbefa4fa4.$

Эти значения представляют собой первые 64 бита дробных частей квадратных корней из восьми простых чисел.

Для SHA-512 инициализируем хэш-переменные $H^{(0)}$ значениями, состоящими из восьми 64-битных слов, которые в hex выглядят так:

$H_0^{(0)} = 6a09e667f3bcc908,$
$H_1^{(0)} = bb67ae8584caa73b,$
$H_2^{(0)} = 3c6ef372fe94f82b,$
$H_3^{(0)} = a54ff53a5fld36fl,$
$H_4^{(0)} = 510e527fade682dl,$
$H_5^{(0)} = 9b05688c2b3e6clf,$
$H_6^{(0)} = 1f83d9abfb41bd6b,$
$H_7^{(0)} = 5be0cdl9137e2179.$

Эти значения представляют собой первые 64 бита дробных частей квадратных корней из восьми простых чисел.

3.5.1 Подсчет хэш-суммы SHA-256

При подсчете хэш-суммы SHA-256 используется ряд дополнительных переменных $W_0, W_1, \ldots, W_{63}$, восемь рабочих переменных *a, b, c, d, e, f, g, h,* переменные для хранения хэш-слов $H_1^{(i)}, H_2^{(i)}, H_3^{(i)}, H_4^{(i)}, H_5^{(i)}, H_6^{(i)}, H_7^{(i)}$. SHA-256 также использует две временные переменные T_1 и T_2, при подсчете применяется операция сложения по модулю 2^{32} (+).

После завершения предварительной обработки каждый из блоков сообщения $M^{(1)}, M^{(2)}, \ldots, M^{(N)}$ участвует в подсчете хэш-суммы по алгоритму, приведенному ниже.

for i = 1 *to N:*

{ 1. Подготовка переменных $\{W_t\}$

$$W_t = \begin{cases} M_t^{(i)} & 0 \le t \le 15 \\ \sigma_1^{(256)}(W_{t-2}) + W_{t-7} + \sigma_0^{(256)}(W_{t-15}) + W_{t-16} & 16 \le t \le 63 \end{cases},$$

2. Инициализируем восемь рабочих переменных *a, b, c, d, e, f, g, h* значениями *i-1* хэш-переменных:

$a = H_0^{(i-1)},$
$b = H_1^{(i-1)},$
$c = H_2^{(i-1)},$
$d = H_3^{(i-1)},$
$e = H_4^{(i-1)},$
$f = H_5^{(i-1)},$

$g = H_6^{(i-1)}$,
$h = H_7^{(i-1)}$.

3. *for* $t = 0$ *to* 63:
{

$$T_1 = h + \sum_1^{(256)}(e) + Ch(e,f,g) + K_t^{(256)} + W_t\ ,$$
$$T_2 = \sum_0^{(256)}(a) + Maj(a,b,c)\ ,$$
$h = g$,
$g = f$,
$e = d + T_1$,
$d = c$,
$c = b$,
$b = a$,
$a = T_1 + T_2$.

}

4. Подсчет промежуточных i-х хэш-значений $H^{(i)}$:
$H_0^{(i)} = a + H_0^{(i-1)}$,
$H_1^{(i)} = b + H_1^{(i-1)}$,
$H_2^{(i)} = c + H_2^{(i-1)}$,
$H_3^{(i)} = d + H_3^{(i-1)}$,
$H_4^{(i)} = e + H_4^{(i-1)}$,
$H_5^{(i)} = f + H_5^{(i-1)}$,
$H_6^{(i)} = g + H_6^{(i-1)}$,
$H_7^{(i)} = h + H_7^{(i-1)}$.

}

После повторения основного цикла N раз (после прохода по каждому $M^{(i)}$ блоку сообщения M) на выходе будем иметь 256-битную последовательность хэш-суммы сообщения M:

$$H_0^{(N)} \| H_1^{(N)} \| H_2^{(N)} \| H_3^{(N)} \| H_4^{(N)} \| H_5^{(N)} \| H_6^{(N)} \| H_7^{(N)}\ . \qquad (30)$$

3.5.2 Подсчет хэш-суммы SHA-512

При подсчете хэш-суммы SHA-512 используется ряд дополнительных переменных $W_0, W_1, \ldots, W_{79}$, восемь рабочих переменных *a, b, c, d, e, f, g, h,* переменные для хранения хэш-слов $H_1^{(i)}, H_2^{(i)}, H_3^{(i)}, H_4^{(i)}, H_5^{(i)}, H_6^{(i)}, H_7^{(i)}$. SHA-512 также использует две временные переменные T_1 и T_2, при подсчете применяется операция сложения по модулю 2^{64} (+).

После завершения предварительной обработки каждый из блоков сообщения $M^{(1)}, M^{(2)}, \ldots, M^{(N)}$ участвует в подсчете хэш-суммы по алгоритму, приведенному ниже.

for $i = 1$ *to N:*
{

1. Подготовка переменных $\{W_t\}$

$$W_t = \begin{cases} M_t^{(i)} & 0 \le t \le 15 \\ \sigma_1^{(512)}(W_{t-2}) + W_{t-7} + \sigma_0^{(512)}(W_{t-15}) + W_{t-16} & 16 \le t \le 79 \end{cases},$$

2. Инициализируем восемь рабочих переменных *a, b, c, d, e, f, g, h* значениями *i-1* хэш-переменных:

$a = H_0^{(i-1)}$,
$b = H_1^{(i-1)}$,
$c = H_2^{(i-1)}$,
$d = H_3^{(i-1)}$,
$e = H_4^{(i-1)}$,
$f = H_5^{(i-1)}$,
$g = H_6^{(i-1)}$,
$h = H_7^{(i-1)}$.

3. *for* $t = 0$ *to* 79*:*

{

$T_1 = h + \sum_1^{(512)}(e) + Ch(e, f, g) + K_t^{(512)} + W_t$,
$T_2 = \sum_0^{(512)}(a) + Maj(a, b, c)$,
$h = g$,
$g = f$,
$e = d + T_1$,
$d = c$,
$c = b$,
$b = a$,
$a = T_1 + T_2$.

}

4. Подсчет промежуточных *i*-х хэш-значений $H^{(i)}$:

$H_0^{(i)} = a + H_0^{(i-1)}$,
$H_1^{(i)} = b + H_1^{(i-1)}$,
$H_2^{(i)} = c + H_2^{(i-1)}$,
$H_3^{(i)} = d + H_3^{(i-1)}$,
$H_4^{(i)} = e + H_4^{(i-1)}$,
$H_5^{(i)} = f + H_5^{(i-1)}$,
$H_6^{(i)} = g + H_6^{(i-1)}$,
$H_7^{(i)} = h + H_7^{(i-1)}$.

}

После повторения основного цикла *N* раз (после прохода по каждому $M^{(i)}$ блоку сообщения *M*) на выходе будем иметь 512-битную последовательность хэш-суммы сообщения *M*:

$$H_0^{(N)} \| H_1^{(N)} \| H_2^{(N)} \| H_3^{(N)} \| H_4^{(N)} \| H_5^{(N)} \| H_6^{(N)} \| H_7^{(N)} \quad (31)$$

3.5.3 Подсчет хэш-суммы SHA-384

При подсчете хэш-суммы SHA-384 используется тот же самый алгоритм, что и в SHA-512 со следующими двумя исключениями:

1. Начальные значения $H^{(0)}$ инициализируются так, как показано выше.
2. 384-битная последовательность соответствует левым 384 битам хэш-суммы SHA-512.

$$H_0^{(N)} \| H_1^{(N)} \| H_2^{(N)} \| H_3^{(N)} \| H_4^{(N)} \| H_5^{(N)} \quad (32)$$

IV Сравнительная характеристика

На рис. 3, представлена сравнительная характеристика некоторых блочных симметричных шифров, работающих на Pentium III (Linux).

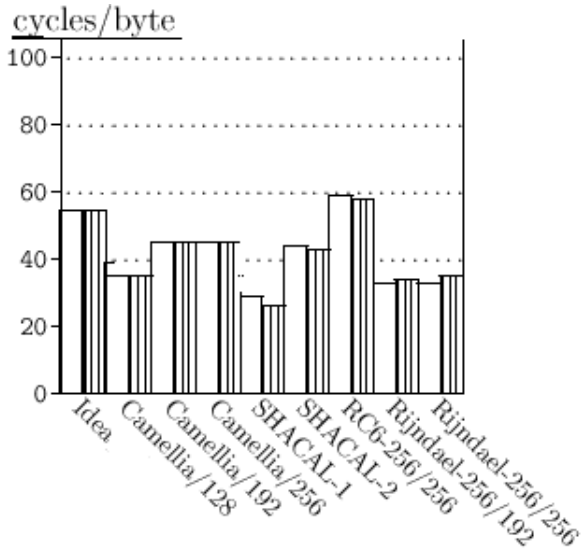


Рисунок 3 – Сравнительная характеристика шифров (Pentium III (Linux))
– зашифрование, – расшифрование.

На рис. 4, представлена сравнительная характеристика некоторых блочных симметричных шифров, работающих на Pentium III (MS Windows).

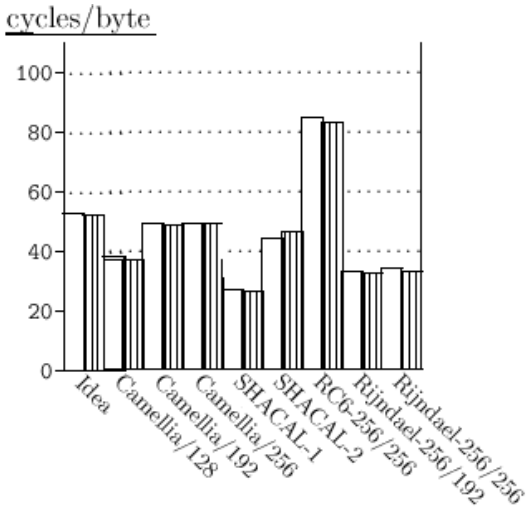


Рисунок 4 – Сравнительная характеристика шифров (Pentium III (MS Windows))
– зашифрование, – расшифрование.

На рис. 5, представлена сравнительная характеристика некоторых блочных симметричных шифров, работающих на Pentium IV.

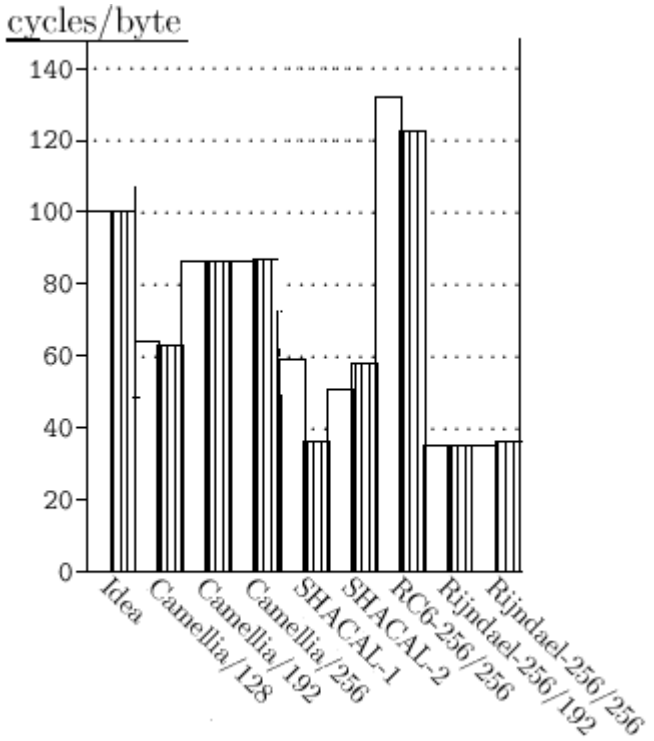


Рисунок 5 – Сравнительная характеристика шифров (Pentium IV)
– зашифрование, – расшифрование.

На рис. 6, представлена сравнительная характеристика работы некоторых хэш-функций работающих на Pentium III.

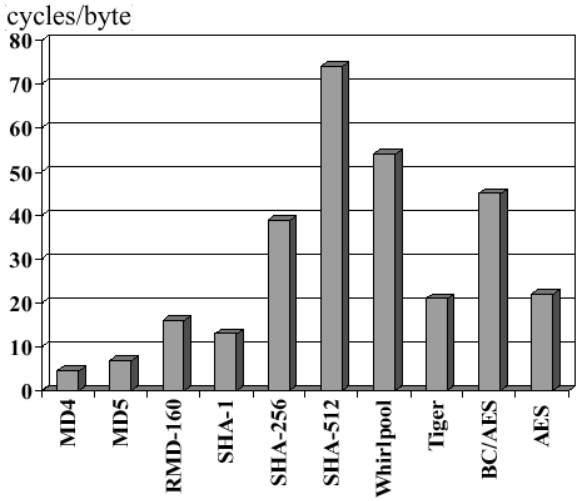


Рисунок 6 – Сравнительная характеристика хэш-функций (Pentium III)

В табл. 1 и 2 приведены скорость программной реализации кандидатов Nessie среди блочных шифров с длиной блока 160, 192 и 256-битов в зависимости от процессора и компилятора.

Таблица 1 – Сравнительные характеристики блочных шифров

Название шифра	Размер ключа	Размер блока	Машина						
			РШ/ Linux Coppemin egcc 2.95.2	РШ/ Linux Coppermine gcc3.1.1	РШ/ Linux Tualatin gcc 2. 96	РШ/ Linux Katmai gcc 3.1.1	РШ/ Linux Coppermine gcc-egcs	РШ/Win Visual C 6.0	РШ/Win Intel C
SHACAL-1	512	160	29/26/78	34/27/445	31/34/515	34/27/75	30/31/439	44/65/839	27/31/715
SHACAL-2	512	256	45/46/745	46/45/737	46/48/891	46/45/871	44/43/793	60/59/1187	46/47/748
RC6 (20 rounds)	256	256	85/89/ 13K	61/58/ 10K	86/85/ 17K	61/58/ 10K	96/88/15 K	120/120/ 17K	125/118/ 12K
Rijndael	192	256	34/39/2426	36/35/1785	37/38/2811	36/34/2001	33/39/2476	33/32/1882	40/39/2068
Rijndael	256	256	33/37/3053	36/36/2298	36/35/3575	36/36/2538	33/37/3146	34/33/2362	35/36/2798

Таблица 2 – Сравнительные характеристики блочных шифров

Название шифра	Размер ключа	Раз - мер блока	Машина						
			PII/Win gcc 2.95.3	PIV/ Linux Northwood gcc 2.95.2	PIV/ Linux Northwood gcc3.1.1	Xeon Model 1 gcc 3.2	AMD Duron WinXP VC7.0 (.NET)	AMD Duron WinXP gcc 2.95.3	AMD Athlon Linux gcc 2.96
SHACAL-1	512	160	30/26/80	63/36/ 177	59/37/ 457	57/37/ 423	37/37/ 54	27/25/ 58	28/24/ 318
SHACAL-2	512	256	44/46/888	55/60/933	51/58/1015	52/59/1075	56/52/861	38/40/597	40/40/633
RC6(20 rounds)	256	256	85/83/12K	190/171/23 K	132/123/21 K	133/120/21K	101/101/18 K	78/79/15K	58/60/12K
Rijndael	192	256	34/39/2558	42/48/4735	35/35/2512	35/35/2557	50/46/2291	50/49/2722	50/50/2430
Rijndael	256	256	34/38/3254	35/44/5982	38/36/3172	34/36/3252	53/47/2985	5256/3561	53/53/3089

Значения в ячейках - показатели времени зашифрования/расшифрования/ настройки ключей, где показатели времени зашифрования/расшифрования указаны в циклах/байт, а показатели времени настройки ключей указаны в циклах на операцию настройки ключа.

V Анализ стойкости

Никаких недостатков безопасности в SHACAL-2 не обнаружено. У него также есть интересное свойство использования большей части хэш-функции SHA-2. SHA-2 – недавно разработанный примитив, поэтому необходимо время для осуществления осторожной и тщательной оценки безопасности SHA-2 и, соответственно, SHACAL-2.

Наилучшей известной атакой в SHA-2 при использовании в качестве хеш-функции является атака Чабода и Джокса. Они показали, что приблизительно за 2^{61} оценок функции сжатия можно найти два сообщения, хэширующихся в одно и то же значение.

Двумя известными наилучшими атаками на системы, подобные SHA в режиме шифрования, являются *линейный криптоанализ* и *дифференциальный криптоанализ*. Существует широкий диапазон вариантов двух атак, предложенных в литературе, но их базовые принципы приблизительно одинаковые. Кроме того, предложено много других атак в схемы шифрования, но они являются менее типичными, чем две вышеупомянутых атаки, и обычно не применяются к схемам шифрования. Эти атаки применяются к SHA только в режиме шифрования, но как мы увидим, сложности атак на базе этих подходов делают их совершенно непрактичными, если вообще возможными.

5.1 Линейный криптоанализ

Линейный криптоанализ пытается идентифицировать серии линейных приближений A_i для различных операционных компонентов в блочном шифре, как, например, *S*-блоки, операции целочисленного сложения, булевы операции или какие-то еще операции. Отдельные линейные приближения затем комбинируются для получения приближения для большей части алгоритма шифрования. Комбинация приближений выполняется путем простой поразрядной операции "Исключающее ИЛИ" так, что конечным приближением будет $A_1 \oplus A_2 \oplus ... \oplus A_n$.

Если линейные приближения A_i имеют место с вероятностью p_i, то тогда определим смещение как $\in_i = | p_i - 1/2 |, \in_i \neq 0$.

При условии, что для каждого $\in_i \neq 0$ приближения A_i они являются возможно полезными в области сложных линейных криптоаналитических атак. После комбинирования обычно оценивается полное смещение с помощью так называемой «леммы о накоплении» (*Piling-Up*) как $\in = 2^{n-1} \prod_{i=0}^{n-1} \in_i$.

Если конечное приближение на всем объеме SHA-1 имеет смещение ε, то требования к данным для атаки задаются с помощью $c \in^{-2} \varepsilon^{-2}$, где c является некоторой константой, зависимой от точного вида атаки. Для рассматриваемых здесь атак с учетом практического опыта предполагается, что $c \approx 8$, но чтобы быть консервативными, мы будем предполагать, что $c = 1$.

Мы упоминали, что нам требуется приближение на большей части шифра. Точно так, как в дифференциальном криптоанализе, существует множество доступных криптоаналитику разнообразных приемов и способов, чтобы избавиться от нескольких избыточных шагов, что возможно позволит восстановить ключевой материал из внешних шагов шифра за такое же время. Число внешних шагов, которые можно удалить таким образом, очень конкретно для используемых приближений и структуры шифра. Однако, как мы увидим, смещения настолько низки с использованием линейного криптоанализа SHA-1, что этот уровень детализации вероятно не более чем отвлечение внимания.

Дифференциальный криптоанализ, включая атаку бумеранга [4] и атаку прямоугольника [5] также применяется к SHACAL-1. Лучшая известная атака работает для 49 шагов функции сжатия со сложностью данных $2^{151,9}$ выбранных открытых текстов и сложностью времени $2^{508,5}$ [5]. Нужно отметить, что вероятность дифференциальных характеристик в SHA-2 уменьшается быстрее чем в SHA-1 . Это является следствием многочисленных вращений в функциях $\sigma_0(x)$ и $\sigma_1(x)$.

5.2 Скользящая атака

Скользящая атака, изобретенная Бирюковым и Вагнером, применима к блочному шифру, где одно и то же значение подключа подается в каждый цикл (или каждый n-й цикл). Эта атака в значительной степени была мотивирована криптоанализом со связанными ключами, введенным Бихэмом. Типичная скользящая атака показана на рис. 7 .

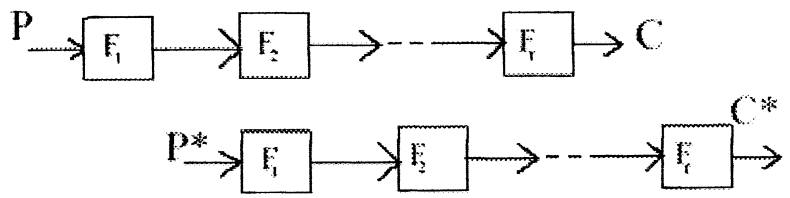


Рисунок 7 – Пример скользящей атаки

Недавно Сааринен [3] отметил, что скользящая атака может быть применена к SHA-1 с успехом приблизительно 2^{32}. Хотя трудно представить себе как эта атака может быть применена для нахождения коллизий или прообразов для хэш-функции, анализ демонстрирует неожиданное свойство функции сжатия. Рассмотрим два сообщения $M = [W_0||W_1||...||W_{15}]$ и $M' = [W'_0||W'_1||...||W'_{15}]$. Главное наблюдение состоит в том, что процедура для расширения сообщения может скользить. Мы просто выбираем $W'_i = W_{i+1}$ для $0 < i < 14$ и $W'_{15} = (W_0 \copyright W_2 \copyright W_8 \copyright W_{13}) <<^1$. После расширения сообщения следующее истинно: $W'_i = W_{i+1}$, для $0 < i < 78$. Второе наблюдение состоит в том, что для 20 шагов в каждом раунде функции сжатия булева функция f_i. и добавочная константа K_i неизменны. Следовательно любые два последовательных шага (шаг i и шаг $i + 1$) функции сжатия подобны, если бы не три перехода между различными раундами (это случается для $i = 19, 39, 59$).

Предположим теперь, что если хэш вычисление для M и M' начинается с начальных значений A, B, C, D, E и A', B', C', D', E' соответственно, которые связаны как $B' = A; C' = B >>^{30}; D' = C; E' = D$. Тогда цель атаки состоит в том, чтобы найти сообщения и начальные значения, для которых одинаковая связь (между переменными формирования цепочки) все еще сохраняется в конце функции сжатия. Такую пару сообщений и передаваемых начальных значений называют "скользящей парой ". Метод для нахождения скользящей пары довольно специфический, но общая стратегия состоит в том, чтобы выбрать подходящие значения для переменных формирования цепочки на шагах $i = 19$ и $i = 39$ и сравнить их. Эта процедура повторяется до перехода $i = 59$, который случается с вероятностью 2^{-32}. Полное время и сложность атаки по размеру имеют порядок 2^{32}.

5.3 Защита от коллизий

Практика показывает, что одной из возможных основных слабостей хэш-функций являются коллизии. Поэтому необходимо уделить внимание защите от коллизии и в построенных на хэш-функциях блочных примитивах SHACAL-1 и SHACAL-2.

Анализ показывает, что в рассматриваемом режиме могут возникать коллизии двух видов - появление идентичных C_i в связи с тем, что используется один и тот же ключ K_j и повторение блоков $M_i = M_{i-v}$. Допустим? что в КРС производится рандомизация. Это позволяет не рассматривать первый тип коллизий.

Если сообщения M_i рандомизованы, то это позволяет рассматривать M_i как равновероятные и независимые реализации. Кроме этого, так как K_j является случайным, то и C_i можно считать равновероятными, независимыми на полном множестве событий. Поэтому будем рассматривать случайные коллизии, при которых в пространстве или времени на выходе преобразователя F с одинаковым ключом будут появляться хотя бы две криптограммы C_i. Используя "парадокс дня рождения", вероятность коллизии можно посчитать как

$$P(n,k) = 1 - e^{-\frac{1}{n} * \frac{k^2 - k}{2}}, \tag{33}$$

где k – число экспериментов,

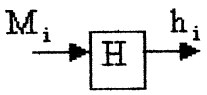
$n = 2^l$ – размер полного множества выхода преобразователя F,

l– длина шифруемого блока.

5.3.1 Парадокс дня рождения

Пусть для группы, состоящей из n человек, необходимо найти вероятность P того, что у двух человек дни рождения совпадут. У 40 человек вероятность $P = 0{,}99$; у 70 человек $P = 0{,}99$.

Пусть алгоритм $H(M) \rightarrow 2^l = n$. Подавая на преобразователь H k сообщений, формируем k хэш-функций. Найти вероятность того, что хотя бы два значения из возможных k хэш-функций совпадут, то есть вероятность того, что произойдет коллизия.



В большинстве источников упоминается, что вероятность коллизии можно найти по формуле $\sqrt{n} = k$ – вероятность того, что при заданных n, k мы найдем P. $Q(n, k)$ – вероятность того, что коллизия не произойдет

$$Q(n,k) = \frac{n(n-1)...(n-k+1)}{n^k}, \tag{34}$$

$$P(n,k) = 1 - Q(n,k), \tag{35}$$

$$P(n,k) = 1 - \frac{n(n-1)...(n-k+1)}{n^k} = 1 - \frac{n}{n}\left(\frac{n-1}{n}\right)\left(\frac{n-2}{n}\right)...\left(\frac{n-k+1}{n}\right) =$$

$$= 1\left(1-\frac{1}{n}\right)\left(1-\frac{2}{n}\right)...\left(1-\frac{k-1}{n}\right), \tag{36}$$

$$k << n,\ \frac{1}{n} < 1,\ \frac{k-1}{n} < 1.$$

Можно доказать, что $1 - x \le e^{-x}$, $x < 1$. Сделаем замену:

$$1 - e^{-\frac{1}{n}} - e^{-\frac{2}{n}}...e^{-\frac{k-1}{n}} = 1 - e^{-\frac{1}{n(1+2+3+...+k-1)}} = 1 - e^{-\frac{1}{n}*\frac{k(k-1)}{2}}, \tag{37}$$

$$P(n,k) = 1 - e^{-\frac{1}{n}*\frac{k^2-k}{2}}, \tag{38}$$

$$e^{\frac{-k^2+k}{2n}} = 1 - P(n,k), \tag{39}$$

$$-k^2 + k = 2n\ln(1-P), \tag{40}$$

$$k^2 - k + 2n\ln(1-P) = 0. \tag{41}$$

Теперь можно найти зависимости между значениями P_k, k и l. Результаты зависимости вероятности P_k от значений k и l приведены в табл. 3.

Таблица 3 – Зависимость вероятности коллизии от длины блока и числа экспериментов

l	k								
	2	2^8	2^{16}	2^{32}	2^{64}	2^{96}	2^{128}	2^{192}	2^{256}
128	0	0	$2{,}2 * 10^{-29}$	$9{,}9 * 10^{-20}$	0,46	~1	~1	~1	~1
160	0	0	0	$6{,}31 * 10^{-30}$	$1{,}16 * 10^{-10}$	~1	~1	~1	~1
256	0	0	0	0	0	$2{,}37 * 10^{-20}$	0,46	~1	~1

Анализ данных табл. 3 показывает, что вероятность коллизии существенно зависит от длины блока l, и длины блоков должны составлять не менее 128 битов. Этот вывод позволяет понять, почему в проекте Nessie для стойких шифров длина блока должна составлять не менее 128 битов.

В табл. 4 приведены значения величин k в зависимости от длины блока l и допустимых вероятностей коллизий P_k. Данные, приведенные в этой таблице, позволяют понять причины и необходимость увеличения в перспективных блочных симметричных шифрах длины блока l.

Таблица 4. – Зависимость числа экспериментов от длины блока и допустимых вероятностей.

l	P_k							
	10^{24}	10^{-16}	10^{-12}	10^{-9}	10^{-6}	10^{-3}	10^{-1}	0,5
128	$8{,}2 * 10^{7}$	$8{,}2 * 10^{11}$	$8{,}2 * 10^{13}$	$2{,}6 * 10^{15}$	$8{,}2 * 10^{16}$	$8{,}2 * 10^{17}$	$8{,}4 * 10^{18}$	$2{,}1 * 10^{19}$
160	$1{,}7 * 10^{12}$	$1{,}7 * 10^{16}$	$1{,}7 * 10^{18}$	$5{,}4 * 10^{19}$	$1{,}7 * 10^{21}$	$5{,}4 * 10^{22}$	$5{,}6 * 10^{23}$	$1{,}5 * 10^{24}$
256	$4{,}8 * 10^{26}$	$5{,}1 * 10^{30}$	$4{,}8 * 10^{32}$	$1{,}5 * 10^{34}$	$4{,}8 * 10^{35}$	$1{,}5 * 10^{37}$	$1{,}5 * 10^{38}$	$4{,}0 * 10^{38}$

Проанализировав результаты, полученные из таблиц 3 и 4, сделаем вывод, что стойкость блочных симметричных шифров напрямую зависит от длины блока. Алгоритмы SHACAL-1 и тем более SHACAL-2 полностью соответствуют по длине блока требованиям сегодняшнего дня.

VI Выводы

Проведя детальный анализ по стойкости SHACAL к наиболее мощным известным сегодня атакам, мы пришли к заключению, что линейная криптоаналитическая атака на SHA как функцию шифрования потребовала бы как минимум 2^{80} известных открытых текстов, а дифференциальная атака – как минимум 2^{116} выбранных открытых текстов. Заметим, что мы подробно рассмотрели конструктивные линейные приближения и дифференциальные характеристики. Очень может быть, что существуют другие приближения и характеристики на SHA-1, которые не обнаруживаются таким типом анализа. Вместо этого, они могли бы быть найдены с помощью атаки «грубой силы». Так как не существует известных приемов для такого поиска, то такая возможность должна рассматриваться как маловероятная настолько, что не представляет практического интереса.

Наши способы построения приближений и характеристик являются специальными, но базирующимися на значительном практическом опыте. Мы были очень осторожными в наших оценках и с большой вероятностью утверждали, что линейная или дифференциальная криптоаналитическая атака с использованием менее чем 2^{80} блоков открытого текста является невозможной. Мы отметили, что с этого места 160-битового блочного шифра начинается утечка информации открытого текста при использовании его для шифрования такого большого текста с тем же ключом.

И, наконец, мы упоминаем, что дополнительные криптоаналитические приемы, такие как линейные оболочки (hulls), многократные линейные приближения и разнообразные виды дифференциалов не способны внести какие-либо существенные отличия в наш анализ и оценки. И поэтому они не представляют никакого практического интереса для сделанных нами выводов.

Заключение

Для SHA-1 была обнаружена атака скольжения, что демонстрирует неожиданное свойство функции сжатия, но это — не угроза для нормального использования хэш-функции. Тем не менее, эта атака также указывает на слабость в ключевом планировании шифровального режима SHA-1. Атака скольжения на SHA-1 не распространяется на SHA-2, поскольку в SHA-2 в каждом шаге функции сжатия используется уникальная добавочная константа. Поэтому проект Nessie и выбрал алгоритм SHACAL-2 победителем и рекомендовал его для всестороннего применения в области криптографической защиты информации.

Литература: ***1.*** *National Institute of Standards and Technology, "New secure hash algorithms." 2000.* ***2.*** *National Institute of Standards and Technology, "FIPS-180-2: Secure Hash Standard (SHS)." Aug. 2002. Available at http://csrc.nist.gov/publications/jips/. [p. 76, 132, 137, 141, 142, 144]* ***3.*** *M-J. O. Saarinen, "Cryptanalysis of block ciphers based on SHA-1 and MDS. "In Proceedings of Fast Software Encryption {FSE'03 (T. Johansson, ed.), Lecture Notes in Computer Science, Springer-Verlag, 2003. Also available at http://www.tcs.hut.fi/~mjos/shaan.ps. [p. 75, 76, 77, 141]* ***4.*** *J. Kim, D. Moon, W. Lee, S. Hong, S. Lee, and S. Jung, "Amplified boomerang attack against reduced-round SHACAL." in Proceedings ofAsiacrypt'Q2 (Y. Zheng,ed.)$_f$ no. 2501 in Lecture Notes in Computer Science, pp. 243 (253, Springer-Verlag, 2002. Also in Proceedings of the Third NESSIE Workshop, 2002. [p. 76, 98, 140, 141]* ***5.*** *E. Biham, N. Keller, and O. Dunkelman, "Rectangle attacks on SHACAL-L "In Proceedings of Fast Software Encryption {FSE'03 (T. Johansson, ed.), Lecture Notes in Computer Science, Springer-Verlag, 2003. NES/DOC/TEC/WP5/031.fp. 76, 98, 141]*

УДК 621.391

ПРИМЕНЕНИЕ СРЕДСТВ КРИПТОГРАФИЧЕСКОЙ ЗАЩИТЫ ИНФОРМАЦИИ В СФЕРЕ БАНКОВСКИХ ТЕХНОЛОГИЙ

Вячеслав Татьянин
ООО «АВТОР»

Аннотация: **Даны рекомендации для правильного выбора систем защиты информации в сфере банковских технологий. Приведены примеры применения средств криптографической защиты информации.**