Andreev Artem Viktorovich – e-mail: andreev.artem@gmail.com; the department of security in data processing technologies; senior programmer.

Mordvin Denis Valerievich – e-mail: neverminden@gmail.com; the department of security in data processing technologies; cand. of eng. sc.; associate professor.

УДК 681.03.245

Л.К. Бабенко, А.С. Кириллов

РАЗРАБОТКА И ИССЛЕДОВАНИЕ АЛГОРИТМОВ АТАКИ НА ГОСТ Р34.11-94 С ИСПОЛЬЗОВАНИЕМ МНОГОПРОЦЕССОРНОЙ СИСТЕМЫ

Описываются особенности реализации алгоритмов атаки на функцию хеширования ГОСТ а также результаты исследований реализованных алгоритмов на предмет возможности полного решения задачи атаки на ГОСТ Р34.11-94. В результате исследований, было выяснено, что существующие алгоритмы не позволяют провести полную атаку на ГОСТ за приемлемое время, о чем свидетельствуют оценки, полученные на основании проведенных экспериментов, представленные в данной работе. Реализованные алгоритмы могут применятся с большим успехом для атаки на другие хеш-функции.

ГОСТ; функция хеширования; параллельные вычисления; атака прообраза; мультиколлизии.

L.K. Babenko, A.S. Kirillov

DEVELOPMENT AND ANALYSIS OF ALGORITHMS OF ATTACK ON THE GOST R34.11-94 USING MULTIPROCESSOR SYSTEM

This article describes features of implementation of algorithms of attack on the GOST hash function, and results of it's analysis concerning the possibility of solution task of GOST R34.11-94 attack. As a result of investigations it was found that the existing algorithms do not allow a full attack on the GOST in acceptable period of time, as indicated by estimates obtained on the basis of the experiments presented in this paper. Implemented algorithms can be applied with great success for attack on other hash functions.

GOST; hash function; parallel computing; preimage attack; multicollision.

Функции хеширования являются одним из основных методов криптографической защиты информации и используются в алгоритмах цифровой подписи, в прикладных системах с открытым ключом [1]. В данной работе задачей ставится реализация алгоритмов атаки на ГОСТ Р34.11-94, исследования их характеристик, возможности их параллельного выполнения, а также определение возможностей современных вычислительных ресурсов для реализации атаки прообраза на ГОСТ.

На сегодняшний день существует 2 основных типа атак на криптографические функции хеширования[2]:

1. Атака нахождения коллизии. Данная атака представляет собой нахождение такой пары M_1 и M_2, хеш-значение которых одинаково и сложность выполнения атаки меньше чем $2^{n/2}$, где n – длина хеш-значения в битах [2].
2. Атака нахождения прообраза [3] криптографической хеш-функции. Данная атака состоит в нахождении сообщения с заданным значением хеша. Существует два типа подобных атак [3]:
   - атака нахождения первого прообраза: по заданному значению хеш h найти такое сообщение M, что $H(M)=h$, где H – функция хеширования, сложность атаки не должна превышать 2^n;

- атака нахождения второго прообраза (атака псевдопрообраза): по данному сообщению M_1 найти такое сообщение M_2, что $H(M_1)= H(M_2)$, сложность атаки не должна превышать 2^n.

Наибольшее развитие и популярность получила атака нахождения коллизий, атака прообраза на сегодняшний день освещена в литературе и публикациях гораздо меньше. Причиной тому, по мнению авторов, может служит меньшая трудоемкость реализации данного вида атаки, по сравнению с атакой прообраза.

В работах [5,6] предложен метод полной атаки на ГОСТ с целью получения, на основании имеющегося хеш-значения, такого сообщения, которое приведет к такому же значению хеш. Предложенный метод состоит из 4-х этапов, с теоретическим описанием каждого, а также с теоретическими оценками сложности для всей атаки и для каждого этапа.

В данной работе будет рассматриваться построение и анализ алгоритмов для реализации описанного метода, в частности для реализации атаки нахождения первого и второго прообразов для ГОСТ Р34.11-94. Как составная часть атаки первого прообраза будет реализован алгоритм атаки построения мультиколлизий.

Перед подробным описанием атак следует рассмотреть сам алгоритм хеширования ГОСТ Р34.11-94.

Краткое описание функции хеширования ГОСТ Р 34.11-94. Для начала приведем основные характеристики данного алгоритма:

- обрабатывается блок длиной 256 бит, и выходное значение тоже имеет длину 256 бит;
- обработка блоков происходит по алгоритму шифрования ГОСТ 28147-89, который содержит нелинейные преобразования на S-блоках, что существенно улучшает криптостойкость алгоритма;
- алгоритм является итеративным.

Рассмотрим процесс хеширования подробнее.

Входное сообщение M разбивается на блоки $m_1, m_2, \dots, m_t$, по 256 бит, в случае, если размер сообщения не кратен 256 битам, выполняется выравнивание сообщения путем добавления нужного количества нулевых разрядов. Далее каждый блок сообщения подается на функцию сжатия для вычисления промежуточного хеш-значения (рис. 1):

$$H_0 = IV, \tag{1}$$

$$H_i = f(H_{i-1}, m_i), \text{ для } 0 < i \le t-1, \tag{2}$$

$$H_{t+1} = f(H_t, m_t), \tag{3}$$

$$H_{t+2} = f(H_{t+1}, |M|), \tag{4}$$

$$H_{t+3} = f(H_{t+2}, \Sigma) = h, \tag{5}$$

где $|M|$ – длина сообщения в битах до выравнивания;

$\Sigma = m_1 \boxplus m_2 \boxplus \dots \boxplus m_t$;

$\boxplus$ – сложение по модулю 256.

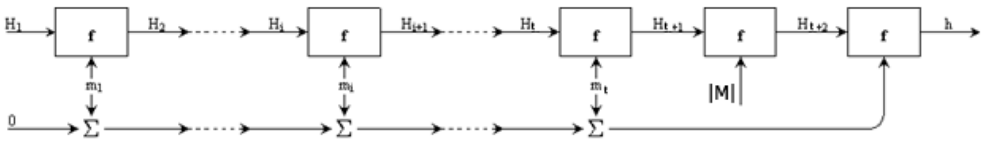

Рис. 1. Структура функции хеширования ГОСТ Р 34.11-94

Основным компонентом данного алгоритма является функция сжатия f, структура которой представлена на рис. 2.

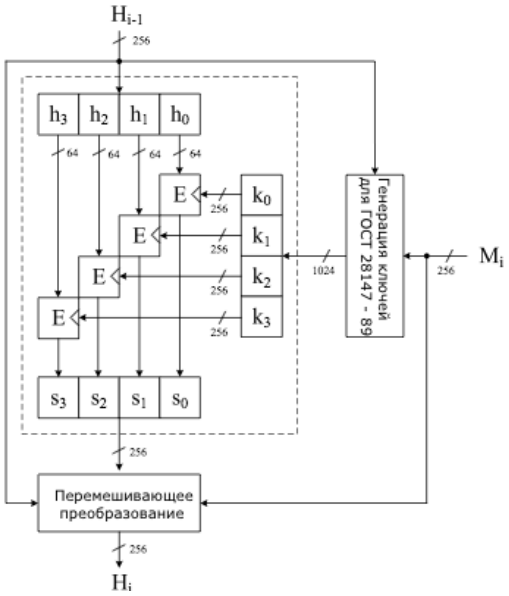

Рис. 2. Структура функции сжатия ГОСТ Р34.11 – 94

Функция сжатия состоит из 4-х параллельных блоков шифрования по ГОСТ 28147-89, который описан в стандарте [4], механизма генерации ключей для выполнения шифрования и выходного перемешивающего преобразования.

На вход функции сжатия поступает хеш-значение, полученное на предыдущей итерации H_{i-1}, которое разбивается на четыре 64-битных слова $h_3 \parallel h_2 \parallel h_1 \parallel h_0$, далее каждое слово шифруется и происходит их объединение $S = s_3 \parallel s_2 \parallel s_1 \parallel s_0$:

$$s_0 = E(k_0, h_0), \quad (6)$$

$$s_1 = E(k_1, h_1), \quad (7)$$

$$s_2 = E(k_2, h_2), \quad (8)$$

$$s_3 = E(k_3, h_3), \quad (9)$$

где $E(K, P)$ – шифрование 64 бит данных P на 256-битном ключе K.

Для генерации ключей используется хеш-значение, полученное на предыдущей итерации H_{i-1}, и текущий блок сообщения M_i, на их основе вычисляется 1024-битный ключ, который разбивается на слова по 256 бит $K = k_3 \parallel k_2 \parallel k_1 \parallel k_0$, которые используются при выполнении шифрования. Вычисление ключей выполняется следующим образом:

$$k_0 = P(H_{i-1} \oplus M_i), \quad (10)$$

$$k_1 = P\left(A(H_{i-1}) \oplus A^2(M_i)\right), \quad (11)$$

$$k_2 = P\left(A^2(H_{i-1}) \oplus Const \oplus A^4(M_i)\right), \quad (12)$$

$$k_3 = P(A(A^2(H_{i-1}) \oplus Const) \oplus A^6(M_i)), \quad (13)$$

где A и P – линейные преобразования; $Const$ – константа.

В данной работе используется только преобразование P:

P:{01} →{01}. Пусть $X=\xi_{32}\|\xi_{31}\|\ldots\|\xi_1$, тогда

$P(X)=\xi_{\varphi(32)}\|\xi_{\varphi(31)}\|\ldots\|\xi_{\varphi(1)}$,

где $\varphi(i+1+4(k-1)) = 8i+k$, $i = 0,3$, $k = 1,8$.

Линейные преобразование A описано в [4].

Выходное перемешивающее преобразование(14) в качестве входных данных использует хеш-значение, полученное на предыдущей итерации H_{i-1}, текущий блок сообщения M_i, а также полученное в результате выполнения шифрования значение S:

$$H_i = \psi^{61}\left(H_{i-1} \oplus \psi\left(M_i \oplus \psi^{12}(S)\right)\right), \quad (14)$$

где ψ – элементарное линейное преобразование.

$$\psi(\mathrm{Y}) = (\mathrm{y}_0 \oplus \mathrm{y}_1 \oplus \mathrm{y}_2 \oplus \mathrm{y}_3 \oplus \mathrm{y}_{12} \oplus \mathrm{y}_{15}) \parallel \mathrm{y}_{15} \parallel \mathrm{y}_{14} \parallel \cdots \parallel \mathrm{y}_1, \quad (15)$$

где $\mathrm{Y} = \mathrm{y}_{15} \parallel \mathrm{y}_{14} \parallel \cdots \parallel \mathrm{y}_0$, конкатенация шестнадцати 16-битных слов.

Для более подробного описания алгоритма ГОСТ Р34.11-94 следует обратиться к [4].

В построении атаки первого прообраза для ГОСТ есть два ключевых момента, это построение атаки псевдопрообраза (или атаки второго прообраза) и построение мультиколлизий. Перед тем как перейти к описанию атаки первого прообраза, рассмотрим данные ключевые моменты подробнее.

Построение атаки псевдопрообраза на функцию сжатия ГОСТ. В данном случае атака основана на уязвимостях в структуре функции сжатия. Основной недостаток – это то, что выходное перемешивающее преобразование ψ – линейная операция, исходя из этого (14) может быть записано как

$$H_i = \psi^{61}(H_{i-1}) \oplus \psi^{62}(M_i) \oplus \psi^{74}(S). \quad (16)$$

Как было сказано выше, из того что ψ линейна, следует, что она обратима, и, таким образом, (16) можно записать как

$$\psi^{-74}(H_i) = \psi^{-13}(H_{i-1}) \oplus \psi^{-12}(M_i) \oplus S, \quad (17)$$

где $\psi^{-74}(H_i) = X$;

$\psi^{-13}(H_{i-1}) = Y$;

$\psi^{-12}(M_i) = Z$.

Заметим, что Y линейно зависит от H_{i-1} и Z линейно зависит от M_i. В отличие от Z и Y, S зависит от обоих H_{i-1} и M_i.

Разобьем 256-битные слова X, Y, Z обозначенные в (17), на 64-битные слова:

$$X = x_3 \parallel x_2 \parallel x_1 \parallel x_0, \quad Y = y_3 \parallel y_2 \parallel y_1 \parallel y_0, \quad Z = z_3 \parallel z_2 \parallel z_1 \parallel z_0.$$

Теперь (16) может быть записано как

$$x_0 = y_0 \oplus z_0 \oplus s_0, \quad (18)$$

$$x_1 = y_1 \oplus z_1 \oplus s_1, \quad (19)$$

$$x_2 = y_2 \oplus z_2 \oplus s_2, \quad (20)$$

$$x_3 = y_3 \oplus z_3 \oplus s_3. \quad (21)$$

Для данного (известного) H_i мы можем легко вычислить значение $X = \psi^{-74}(H_i)$. Теперь допустим, что для $X = x_3 \parallel x_2 \parallel x_1 \parallel x_0$ можно найти две пары (H_{i-1}^1, M_i^1) и (H_{i-1}^2, M_i^2), где $H_{i-1}^1 \neq H_{i-1}^2$ или $M_i^1 \neq M_i^2$, такие что обе пары порождают одинаковое x_0. Теперь мы знаем, что с вероятностью 2^{-192} эти две пары к тому же приводят к соответствующим значениям x_1, x_2 и x_3. Другими словами, мы имеет построенный псевдопрообраз для данного H_i для функции сжатия ГОСТ с вероятностью 2^{-192}. Поэтому, если мы сможем получить 2^{192} пар (H_{i-1}^j, M_i^j), где $H_{i-1}^j \neq H_{i-1}^j$, или $M_i^j \neq M_i^j$, которые все дают одинаковое значение x_0, тогда мы построим псевдопрообраз для функции сжатия ГОСТ [5].

Рассмотрим этот процесс подробнее. Сначала выясним, как получить пары (H_{i-1}^j, M_i^j), которые все дают одинаковое значение x_0. Это сводится к решению неопределенной системы уравнений. Допустим, мы принимаем s_0 в (18) как константу. Теперь $s_0 = E(k_0, h_0)$, и мы должны найти пары (H_{i-1}^j, M_i^j), для которых значения k_0 и h_0 одинаковые. Мы знаем, что h_0 напрямую зависит от H_{i-1}. Ключ k_0 зависит от $H_{i-1} \oplus M_i$. Следовательно, мы получаем следующие уравнения:

$$h_0 = a; \quad (22)$$

$$m_0 \oplus h_0 = b_0; \quad (23)$$

$$m_1 \oplus h_1 = b_1; \quad (24)$$

$$m_2 \oplus h_2 = b_2; \quad (25)$$
$$m_3 \oplus h_3 = b_3, \quad (26)$$

где a и b_i – любые 64-битные числа.

Заметим, что $k_0 = P(H_{i-1} \oplus M_i) = \bar{B}$, где $\bar{B} = P(B)$ и $B = b_3 \parallel b_2 \parallel b_1 \parallel b_0$(см (10)). Это неопределенная система в $GF(2)$. Решение этой системы приводит к 2^{192} вариантам, для которых s_0 имеет одинаковое значение. Для нахождения пар (H_{i-1}^j, M_i^j), у которых x_0 имеет одинаковое значение, также мы должны гарантировать, что элемент $x_0 \oplus y_0$ в (18) имеет одинаковое значение для всех. Это добавляет одно дополнительное уравнение в нашу систему, а именно

$$y_0 \oplus z_0 = c, \quad (27)$$

где c – произвольное 64-битное значение.

Это уравнение не добавляет новых переменных, так как мы знаем, что y_0 линейно зависит от $h_3 \parallel h_2 \parallel h_1 \parallel h_0$ и z_0 зависит линейно от $m_3 \parallel m_2 \parallel m_1 \parallel m_0$ (см. 17). Установка значения x_0 приводит к решению описанной неопределенной системы уравнений над $GF(2)$. Это приводит к 2^{128} решений h_i и m_i для $0 \le i < 4$ и, таким образом, к 2^{128} пар (H_{i-1}^j, M_i^j), для которых x_0 имеет одинаковое значение [6].

Теперь мы можем описать, как работает атака псевдопрообраза на ГОСТ. В этой атаке мы должны найти H_{i-1} и M_i, такие что $f(H_{i-1}, M_i) = H_i$ для данного H_i.

Алгоритм 1.

Вход: H_i – значение хеш интересующей нас итерации.

Выход: Пары H_{i-1} и M_i, такие что $f(H_{i-1}, M_i) = H_i$.

1. Случайно выбрать числа b_0, b_1, b_2, b_3 и a (они определяют k_0, h_0).

2. Вычислить $s_0 = E(k_0, h_0)$ и подобрать согласно этому выражению:

$$x_0 = y_0 \oplus z_0 \oplus s_0 = c \oplus s_0,$$

где $X = \psi^{-74}(H_i)$.

3. Решить приведенную выше систему уравнений для получения 2^{128} пар (H_{i-1}^j, M_i^j), для которых x_0 соответствует значению, полученному на 2-м этапе.

4. Для каждой пары вычислить X и проверить соответствие x_1, x_2, x_3 входному значению H_i. Это выполняется с вероятностью 2^{-192}. Таким образом, после проверки всех 2^{128} пар мы найдем верную пару с вероятностью $2^{-192} * 2^{128} = 2^{-64}$.

5. Вернуться к п. 1. (Повторить атаку 2^{64} раз для разных b_0, b_1, b_2, b_3 и a для нахождения псевдопрообраза для функции сжатия ГОСТ).

Таким образом, мы можем построить псевдопрообраз для функции сжатия ГОСТ со сложностью около 2^{192} вместо 2^{256}.

Описание алгоритма построения мультиколлизий. В качестве исходных данных считаем, что имеется механизм для нахождения коллизий, обозначим его как C, в качестве входных данных которого выступает значение h, а выходными данными являются два блока B и B', такие что $f(h, B) = f(h, B')$. Данный механизм может использовать как метод парадокса дней рождений, так и другой способ нахождения коллизий, основанный на уязвимости функции сжатия f. Наиболее существенным свойством, которым должно обладать C, является работоспособность для любого значения h. Для того чтобы продемонстрировать суть идеи, покажем, как может быть найдено 4 коллизии в результате двух вызовов C. Сначала сгенерируем инициализирующий вектор IV, используемый при первом вызове C, далее, выполнив один вызов C, получим 2 различных блока B_0 и B_0', которые приводят к коллизии, т.е. $f(IV, B_0) = f(IV, B_0')$. Обозначим z как их общее значение $f(IV, B_0) = f(IV, B_0') = z$ и будем его использовать при следующем вызове C в качестве входного параметра, для нахождения следующих блоков B_1 и B_1', таких что $f(z, B_1) = f(z, B_1')$. Объединив два описанных выше шага, получим следующие 4 коллизии:

$$f(f(IV, B_0), B_1) = f(f(IV, B_0), B_1') = f(f(IV, B_0'), B_1) =$$
$$= f(f(IV, B_0'), B_1').$$

На основании описанной базовой идеи можно сформировать алгоритм для нахождения значительно большего количества коллизий, используя вызовы механизма С. Таким образом, выполнив t вызовов С, можно осуществить построение 2^t коллизий [7].

Алгоритм 2.

Вход: Инициализирующий вектор IV, значение t, равное желаемому количеству коллизий, исходя из того, что количество коллизий равно 2^t.

Выход: 2^tсообщений в форме $b_1, \ldots, b_t$, где b_t – один из двух блоков B_i или B_i'.

1. Установить $h_0 = IV$.
2. Присвоить $i = 1$, цикл по i пока $i \le t$.
   2.1. Выполнить вызов С и найти B_0 и B_i', такие что $f(h_{i-1}, B_i) = f(h_{i-1}, B_i')$.
   2.2. Установить $h_i = f(h_{i-1}, B_i)$.

Очевидно, что 2^t различных сообщений построены таким образом, что конечное значение h будет одинаковым, а именно все промежуточные значения h одинаковы. На рис. 3 приведено графическое представление работы описанного алгоритма.

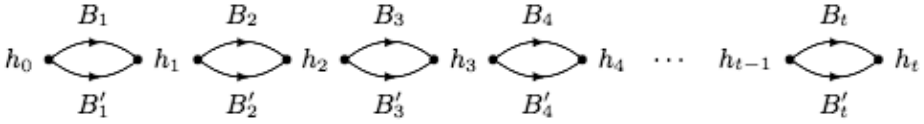

Рис. 3. Графическое представление работы алгоритма, осуществляющего построение мультиколлизий

На основании вышеописанного перейдем к рассмотрению атаки прообраза для функции хеширования ГОСТ.

Построение атаки прообраза для функции хеширования ГОСТ.

Как было сказано выше, в данной атаке мы хотим найти для данного значения хеш h такое сообщение M, что $H(m) = h$. В дальнейшем будет показано, что мы можем сконструировать прообраз h с вычислительной сложностью 2^{255} вычислений функции сжатия ГОСТ. Прообраз, построение которого будет осуществляться состоит из 257 блоков сообщения, т.е. $M = M_1 \| \ldots \| M_{257}$. Атака включает в себя 4 этапа представленных на рис. 4 в виде разделений пунктирными линиями.

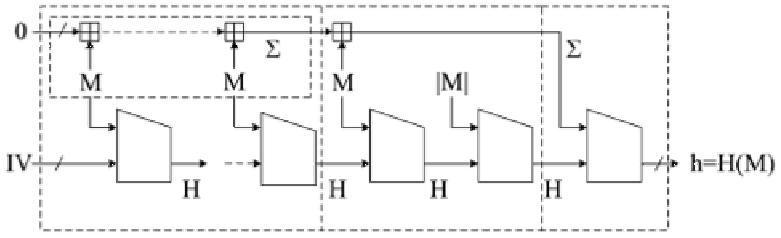

Рис. 4. Построение атаки прообраза для функции хеширования ГОСТ

1-й этап

В предыдущем разделе был описан алгоритм построения мультиколлизий. Для атаки первого прообраза на ГОСТ мы должны получить 2^{256}мультиколизий. Это означает, что мы имеем 2^{256} сообщений $M^* = M_1^{j1} \| M_2^{j2} \| \cdots \| M_{256}^{j256}$ для

$j_1, j_2, \dots, j_{256} \in \{1, 2\}$, состоящих из 256 блоков, которые все дают одинаковое значение хеш H_{256}. Сложность составляет $256 * 2^{128} = 2^{136}$ вычислений хеш-функции, требуется 2^{14} байт памяти.

2-й этап

Выполняется построение 2^{32} псевдопрообразов для последней итерации ГОСТ, пользуясь алгоритмом, приведенным в соответствующем разделе. В результате будет построен список L, содержащий 2^{32} пар $(H_{258}, \sum^t)$. Сложность составляет 2^{224} вычислений хеш, для хранения списка требуется 2^{38} байт памяти.

3-й этап

На этом этапе требуется найти блок сообщения M_{257} такой, который для данного H_{256}, определенного на 1-м этапе, и для $|M|$, определенного нашим предположением, что мы хотим сконструировать прообраз, состоящий из 257 блоков сообщения, мы находим H_{258}, который также содержится в списке L, полученном на 2-м этапе. Выполним следующие действия [6]:

Алгоритм 3.

Вход: Значение H_{256}, полученное на 1-м этапе.

Выход: Связующий блок сообщения M_{257}.

1. Выбрать произвольный блок сообщения M_{257} и вычислить H_{258} следующим образом:

$$H_{257} = f(H_{256}, M_{257});$$

$$H_{258} = f(H_{257}, |M|),$$

где $|M| = (256 + 1) * 256$.

2. Проверить, находится ли полученное значение H_{258} в списке L. (Проверив все 2^{32} элементов в L, мы найдем верное M_{257} с вероятностью $2^{-256} * 2^{32} = 2^{-224}$.)

3. Если подходящее значение M_{257} не найдено, перейти к 1-му этапу, если же подходящее значение найдено, запомнить его и выполнить выход.

Таким образом, после повторения этого этапа 2^{224} раза мы найдем M_{257} и соответствующее H_{258}, которое также содержится в списке L. Этот этап требует 2^{225}вычислений хэш. Теперь мы нашли подходящее M_{257}, а также значение $\sum^m$, определенное как $\sum^m = \sum^t \boxminus M_{257}$ [6].

4-й этап

На 1-м этапе нами получены 2^{256}мультиколлизий. Теперь мы находим сообщение $M^* = M_1^{j_1} \| M_2^{j_2} \| \cdots \| M_{256}^{j_{256}}$ для $j_1, j_2, \dots, j_{256} \in \{1, 2\}$, которое позволяет получить $\sum^m = \sum^t \boxminus M_{257}$. Это может быть легко выполнено применением метода встречи посередине [5].

Алгоритм 4.

Вход: $\sum^m = \sum^t \boxminus M_{257}$.

Выход: Цепочка сообщений из списка мультиколлизий.

1. Вычислить и сохранить все $\sum_1 = M_1^{j_1} \boxplus M_2^{j_2} \boxplus \dots \boxplus M_{128}^{j_{128}}$ для каждой цепочки сообщений в списке мультиколлизий в списке L.
2. Вычислить для текущей цепочки сообщение $\sum_2 = M_{129}^{j_{129}} \boxplus M_{130}^{j_{130}} \boxplus \dots \boxplus M_{256}^{j_{256}}$.
3. Проверить, есть ли $\sum^m \boxminus \sum_2$ в списке L.
4. Если соответствия не найдено, перейти к следующей цепочке и ко 2-му этапу.

После тестирования 2^{128}значений мы ожидаем найти нужное соответствие элементов в списке L и, следовательно, нужное сообщение $M^* = M_1^{j_1} \| M_2^{j_2} \| \cdots \| M_{256}^{j_{256}}$,

которое приводит к $\sum^m = \sum^t \boxminus M_{257}$. Таким образом, можно найти прообраз для ГОСТ, состоящий из $256 + 1$ блоков сообщения, $M^* \parallel M_{257}$.

Особенности реализации описанных алгоритмов. При выполнении этапа построения мультиколлизий основной частью является работа механизма C, осуществляющего поиск коллизии. Как было сказано выше, в качестве этого механизма может выступать любой алгоритм, выполняющий поиск коллизий, или более универсальный метод парадокса дней рождений. Для функции хеширования ГОСТ на данный момент нет алгоритма нахождения коллизий для функции сжатия, удовлетворяющего описанному в соответствующем разделе обязательному условию работы механизма C. По этой причине для нахождения коллизии для функции сжатия будет использоваться алгоритм парадокса дней рождения. Согласно формуле, описанной в [8]:

$$n(p, d) \approx \sqrt{2d * \ln\left(\frac{1}{1-p}\right)}, \tag{28}$$

где n – колличество требуемых пар (h, M);

p – требуемая вероятность успешности атаки;

d – количество возможных значений h,

ориентировочный расход памяти на хранение пар (h, M), для вероятности успешности атаки в 50 %, составляет $(4 * 10^{38}) * 128 = 512 * 10^{38}$байт. На данный момент, для современных вычислительных ресурсов, это недостижимо, однако для демонстрации верности идеи было принято решение осуществить поиск коллизии для функции сжатия меньшего размера, в качестве которой была выбрана некриптографическая функция хеширования murmur264 [9] с выходным хеш-значением в 32 бита. Согласно формуле (28) для поиска коллизии данной хеш-функции с вероятностью успеха в 50 % требуется $(5{,}1 * 10^9) * 80 = 408 * 10^9$ байта, что эквивалентно 379 Гб памяти. Это достаточно приемлемая цифра, однако сгенерировать и произвести поиск в таком массиве данных само по себе нетривиальная задача, поэтому авторами был разработан программный комплекс, выполняющий поэтапное генерирование и слияние данных в общий массив с последующим поиском подходящей пары.

Приведем словесное описание алгоритма работы программы:

Алгоритм 5.

Вход: h_{i-1} хеш-значение.

Выход: Пары (h_{i-1}, M_1) и (h_{i-2}, M_2), такие что $H(h_{i-1}, M_1) = H(h_{i-1}, M_2)$.

1. На первом этапе генерировать 2^n , где n – любое натуральное число $n \geq 1$, блоков с данными, в которых содержатся упорядоченные по полю h пары (h, M).
2. Если на данный момент есть только 1 блок, перейти к п. 5.
3. Выполнить сортировку слиянием по полю h каждой пары получившихся на предыдущем этапе блоков в один новый блок.
4. Если блоки, полученные на предыдущем этапе, исчерпаны, перейти к следующему этапу и к п. 2.
5. Выполнить проход по полученному блоку, тем самым произведя поиск одинаковых значений поля *h*.
6. Если поиск завершился успешно, произвести выход и вывод полученных пар, в противном случае перейти к п. 1.

Графическая иллюстрация работы алгоритма приведена на рис. 5.

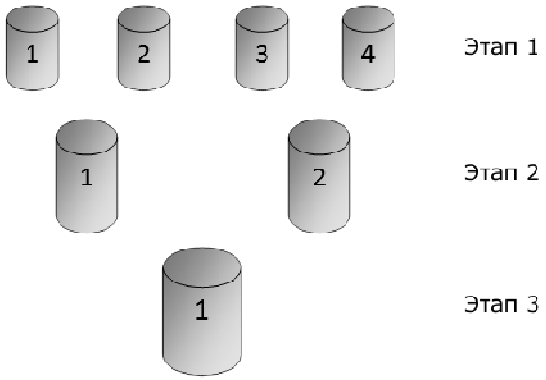

Рис. 5. Графическая иллюстрация работы алгоритма поиска коллизий методом парадокса дней рождений

Разработанный алгоритм является легко масштабируемым и позволяет в зависимости от возможности вычислительной системы, в частности оперативной памяти, получить существенный прирост производительности. Размер оперативной памяти влияет на начальный размер генерируемого блока, таким образом, чем больший объем оперативной памяти имеется в распоряжении вычислительного узла, тем большего размера первоначальный блок данных может быть быстро и эффективно сгенерирован. Разработанный алгоритм позволяет, в случае маленького объема оперативной памяти, достичь нужного размера конечного блока путем увеличения количества первоначально генерируемых блоков. Например, для получения выходного блока данных размером в 100 Гб для машины, обладающей 2 Гб оперативной памяти, ориентировочно (не учитывается память, занимаемая ОС, и память, занимаемая указателями на генерируемые данные) требуется сгенерировать на начальном этапе 50 блоков, соответственно для машины с размером оперативной памяти 4 Гб на начальном этапе требуется сгенерировать 25 блоков. Результаты экспериментов по данному вопросу представлены на графике рис. 6, показывающем зависимость скорости решения задачи от объема используемой ОЗУ.

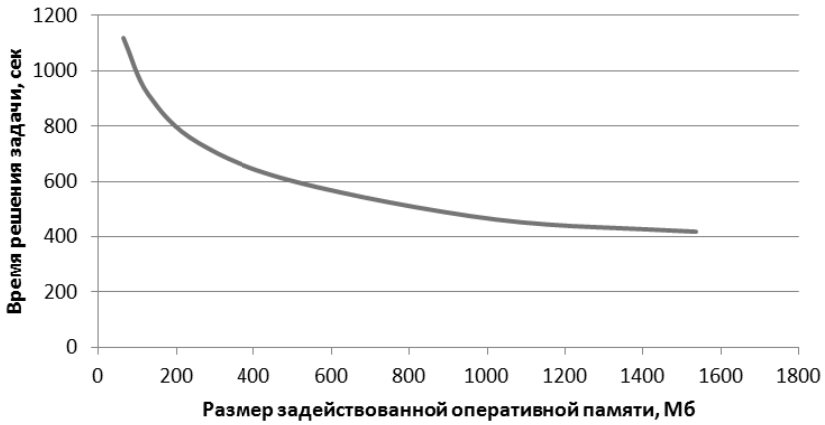

Рис. 6. График зависимости скорости решения задачи от объема используемой ОЗУ

Как уже было сказано выше, объем обрабатываемый данных достаточно велик, соответственно хранение 379 Гб данных в оперативной памяти не представляется возможным, поэтому все блоки данных на каждом этапе хранятся на жестком диске вычислительной машины. Для того чтобы обойти ограничения в скорости работы жесткого диска по сравнению со скоростью работы ОЗУ, в разработанном алгоритме используется система кэширования данных на этапе сортировки блоков данных по алгоритму слияния, путем опережающего чтения устанавливаемого объема данных с диска. Также данный алгоритм можно реализовать и в параллельном исполнении, и, согласно теоретическим расчетам на основании закона Амдала, примерный выигрыш при использовании восьми вычислительных ядер может составить 4,88 раза. Однако после реализации данного алгоритма в параллельном исполнении были получены неожиданные результаты, представленные на рис. 7.

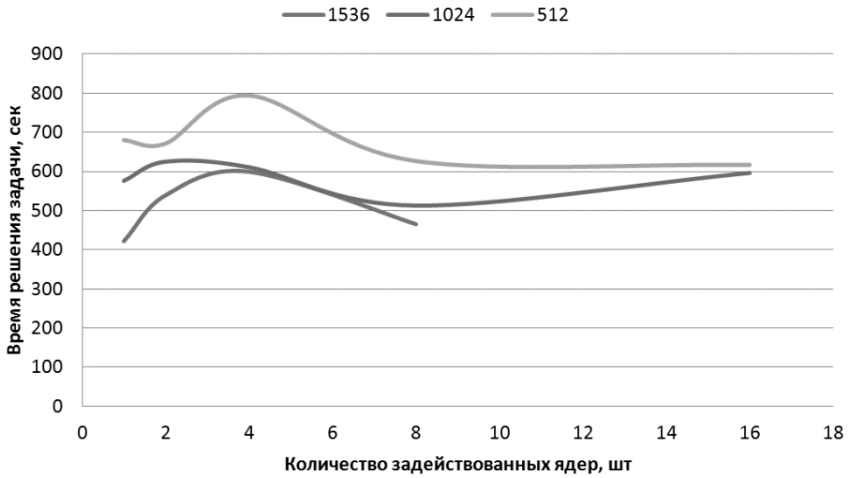

Рис. 7. График зависимости скорости решения задачи от количества используемых вычислительных ядер

На представленном графике характер кривой, соответствующей объему ОЗУ в 1536 Мб, обосновывается следующим образом.

На начальном этапе наблюдается рост времени решения задачи, несмотря на большее количество задействованных ядер. Это связано с тем фактом, что для генерации начальных блоков задействуются сравнительно малые ресурсы, при этом в связи с используемой файловой системой кластера образуется очередь на запись данных, которая и вызывает задержку. В точке, когда в работу включены 8 ядер, наблюдается спад, по причине того, что в данном случае для генерации данных используется достаточно большое количество ресурсов, которое компенсирует задержку записи данных на диск. Далее снова наблюдается рост времени выполнения. Это связано с тем, что очередь на запись возрастает до совершенно неприемлемого размера, что приводит к резкому спаду производительности. В дальнейшем планируется произвести оптимизацию разработанной программы для минимизации полученного эффекта задержки при записи данных на диск.

Наиболее оптимальным количеством ядер для решения задачи (на основании представленного графика) является восемь. Для этого случая и для случая, когда размер задействованной ОЗУ составляется 1536 Мб, на рис. 8 представлен график зависимости скорости решения задачи от размера требуемого конечного блока данных.

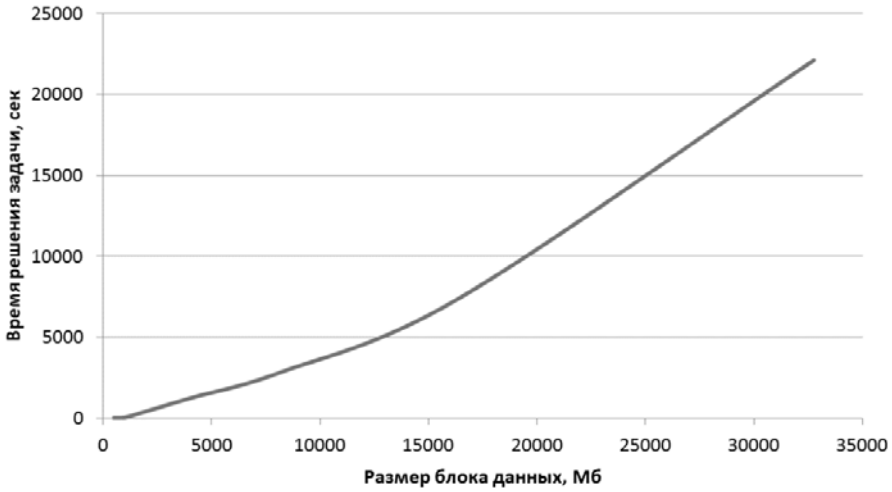

Рис. 8. Скорость решения задачи в зависимости от размера конечного блока данных

Данный алгоритм прошел тестирование на кластере фирмы HP, с пятью вычислительными узлами, в каждом из которых установлено по одному 4-ядерному процессору Intel Xenon с 2 Гб ОЗУ. Для выполнения поставленной задачи, для поиска коллизии согласно описанным выше условиям потребовалось 61,65 часов.

При реализации этапа построения прообраза для последней итерации ГОСТ основным используемым ресурсом является вычислительная мощность процессора. Реализованный алгоритм решения для описанной в разделе построения псевдопрообраза для последней итерации ГОСТ системы уравнений позволяет найти одно значение прообраза за 39 часов. При реализации данного алгоритма в параллельном исполнении следует уделить особое внимание распределению интервалов для вычисления. При этом наиболее эффективным подходом является распределение интервалов для h_3, это позволяет получить начальный результат несколько быстрее.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Мао, Венбо. Современная криптография: теория и практика: Пер. с англ. – М.: Изд. дом "Вильямс", 2005. – 768 с.
2. Cryptographic Hashes [Электронный ресурс] 2009. – Режим доступа: http://www.vpnc.org/hash.html, свободный. – Загл. с экрана.
3. Schneier B., Hoffman P. Attacks on Cryptographic Hashes in Internet Protocols [Электронный ресурс] / P. Hoffman VPN Consortium B. Schneier Counterpane Internet Security, 2005. – Режим доступа: http://tools.ietf.org/html/rfc4270, свободный. – Загл. с экрана.
4. ГОСТ Р 34.11-94. Криптографическая защита информации. Функция хэширования. – М.: Изд-во стандартов, 1994. – 15 с.
5. Florian Mendel, Norbert Pramstaller, and Christian Rechberger. A (Second) Preimage Attack on the GOST Hash Function. In Kaisa Nyberg, editor, FSE, volume 5086 of LNCS. – Springer, 2008. – P. 224-234
6. Florian Mendel, Norbert Pramstaller, Christian Rechberger, Marcin Kontak, Janusz Szmidt. Cryptanalysis of the GOST Hash Function. Advances in Cryptology, CRYPTO 2008: 28th Annual International Conference Santa Barbara, CA, USA, August 2008. Proceedings. Springer, 2008.
7. Antoine Joux. Multicollisions in Iterated Hash Functions. Application to Cascaded Constructions. In Matthew K. Franklin, editor, CRYPTO, volume 3152 of LNCS, Springer, 2004. – P. 306-316.

8. Birthday problem [Электронный ресурс] – Режим доступа: http://en.wikipedia.org/wiki/Birthday_problem, свободный. – Загл. с экрана.
9. Murmur264 [Электронный ресурс] – Режим доступа: http://www.team5150.com/~andrew/noncryptohashzoo/Murmur264.html, свободный. – Загл. с экрана.

Статью рекомендовал к опубликованию д.т.н., профессор Н.И. Витиска.

Бабенко Людмила Климентьевна – Технологический институт федерального государственного автономного образовательного учреждения высшего профессионального образования «Южный федеральный университет» в г. Таганроге; e-mail: blk@fib.tsure.ru; 347928, г. Таганрог, ул. Чехова, 2; тел.: 88634312018; кафедра безопасности информационных технологий; профессор.

Кириллов Алексей Сергеевич – e-mail: kirillovalexeys@gmail.com; кафедра безопасности информационных технологий.

Babenko Lyudmila Klimentevna – Taganrog Institute of Technology – Federal State-Owned Autonomy Educational Establishment of Higher Vocational Education “Southern Federal University”; e-mail: blk@fib.tsure.ru; 2, Chehov Street, Taganrog, 347928, Russia; phone: +78634312018; the department of security of information technologies; professor.

Kirillov Alexey Sergeevich – e-mail: kirillovalexeys@gmail.com; the department of security of information technologies.