Хеш-функции. Теория, применение и новые стандарты (часть 1)

Иванов М.А., Стариковский А.В.

2015 г. (Исправленная версия 2017 г.)

Хеширование являются важным видом криптографического преобразования. Область применения хеш-функций чрезвычайно широка, они успешно решают практически все задачи защиты компьютерной информации от обеспечения аутентичности субъектов и объектов информационного взаимодействия до внесения неопределенности в работу средств и объектов защиты. При этом стоит отметить, что задача проектирования качественной хеш-функции более сложная, чем задача проектирования качественного симметричного шифра.

1. Основы теории

1.1. Определение и принцип действия

Хеш-функция $h(x)$ – это функция, принимающая на входе в качестве аргумента информационную последовательность (строку) M произвольной длины и дающая на выходе в качестве результата информационную последовательность (строку) фиксированной длины. Результат хеширования информационной последовательности M называется *хеш-образом* $h(M)$. Соотношение между длинами M и $h(M)$ может быть произвольным, иначе говоря, возможны любые соотношения

$$|M| > |h(M)|,\ |M| < |h(M)|,\ |M| = |h(M)|,$$

хотя чаще встречается первое, где $|M|$ – длина информационной последовательности M. Так как результат действия хеш-функции называется *хеш-образом*, массив данных M иногда называют *прообразом* (*первым прообразом*).

Дадим формальное определение хеш-функции. Пусть $\{0, 1\}^m$ – множество всех двоичных строк длины m, $\{0, 1\}^*$ – множество всех двоичных строк конечной длины. Тогда хеш-функцией h называется преобразование вида

$$h : \{0, 1\}^* \rightarrow \{0, 1\}^m,$$

где m – разрядность хеш-образа.

На рис. 1 показана схема хеширования, где PRNG – генератор псевдослучайных чисел (Pseudo-Random Number Generator); Q – элементы памяти PRNG; h_0 – вектор инициализации (*IV*); $n = |m_i|$ – разрядность блоков информационной последовательности, $i = 1, \ldots, t$:

$$M = m_1, \ldots, m_t,$$

t – число блоков последовательности M; N – число элементов памяти PRNG. Процесс получения хеш-функции можно упрощенно рассматривать как наложение

псевдослучайной последовательности (PRS, Pseudo-Random Sequence) на входную преобразуемую последовательность. По этой причине иногда спецификация синхронного поточного шифра описывает и родственную хеш-функцию.

Найти *коллизию* хеш-функции, значит, найти два произвольных различных массива M_1 и M_2, таких что $h(M_1) = h(M_2)$. Иначе говоря, для двух разных аргументов значения хеш-функции совпадают. На рис. 2 коллизия возникает при хешировании массивов M_3 и M_4.

Найти *второй прообраз* функции $h(x)$ значит, найти по заданному массиву данных M (первому прообразу) и его хеш-образу $h(M)$ другой массив $M' \neq M$, такой, что $h(M) = h(M')$.

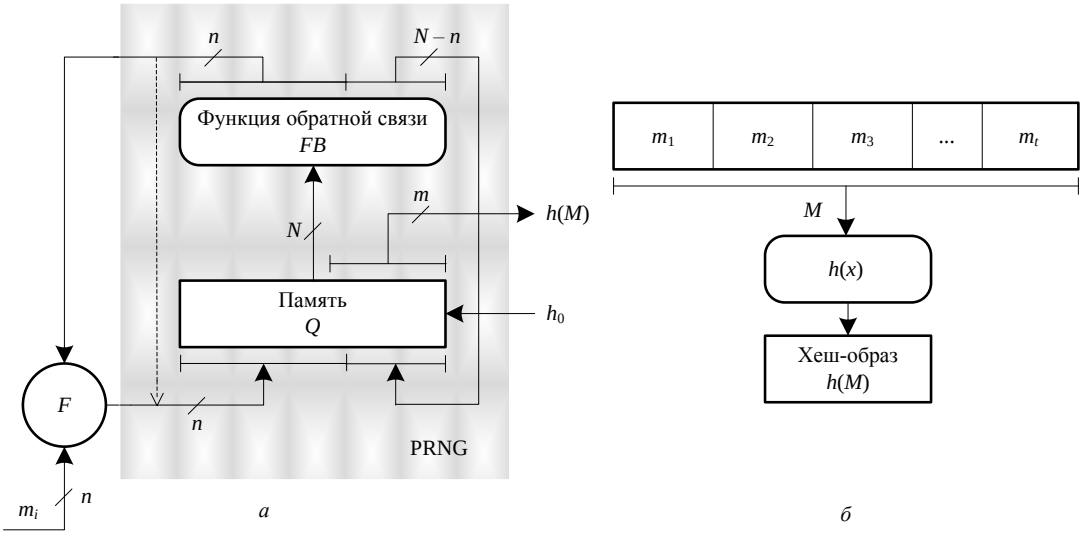

Рисунок 1 – Хеш-функция: *а* – наложение PRS на входную информационную последовательность; *б* – упрощенный принцип действия хеш-функции.

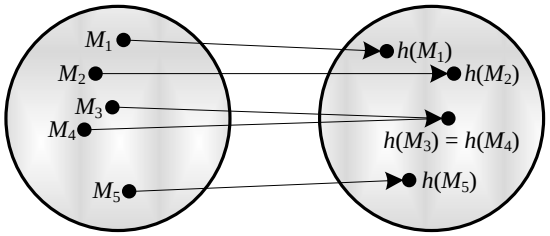

Рисунок 2 – Множества прообразов и хеш-образов.

1.2. Требования к криптографической хеш-функции

Любая криптографическая хеш-функция $h(x)$ должна удовлетворять следующим требованиям [1-4]:

– Хеш-образ должен зависеть от всех бит прообраза и от их взаимного расположения;

– При любом изменении входной информации, хеш-образ должен изменяться непредсказуемо, иначе говоря, в среднем должна измениться половина бит хеш-образа (каждый бит может измениться с вероятностью 1/2);
– При заданном значении прообраза задача нахождения хеш-образа должна быть вычислительно разрешима, иначе говоря, по заданному значению *M* легко вычисляется значение *h*(*M*);
– При заданном значении хеш-образа задача нахождения прообраза должна быть вычислительно неразрешимой, иначе говоря, по заданному значению *h*(*M*) трудно вычислить[1] значение *M* (рис. 3,*а*) (*Pre-Image Resistance*); формально говоря, хеш-функция h является односторонней (*One-Way*), если для произвольной *n*-разрядной строки $y \in \{0,1\}^n$ вычислительно сложно найти $x \in \{0,1\}^*$, такое, что *h*(*x*) = *y*;
– При заданных значениях хеш-образа *h(M)* и первого прообраза *M* задача нахождения второго прообраза *M' ≠ M,* такого, что *h(M')* = *h(M),* должна быть вычислительно неразрешимой (рис. 3,*в*) *(Second Pre-Image Resistance*); формально говоря, хеш-функция h является стойкой в смысле нахождения второго прообраза, если для заданной *m*-разрядной строки $x \in \{0,1\}^m$ вычислительно сложно найти произвольную строку $x' \in \{0,1\}^*$, $x' \neq x$, такую, что $h(x') = h(x)$;
– Задача нахождения коллизии хеш-функции, т.е. нахождение двух произвольных сообщений M_1 и M_2, таких, что $M_1 \neq M_2$, а $h(M_1) = h(M_2)$, должна быть вычислительно неразрешимой (рис. 3,*б*) (*Collision Resistance*); формально говоря, хеш-функция h является стойкой в смысле нахождения коллизий, если вычислительно сложно найти две произвольные строки $x', x'' \in \{0,1\}^*$, $x' \neq x''$, такие, что $h(x') = h(x'')$.

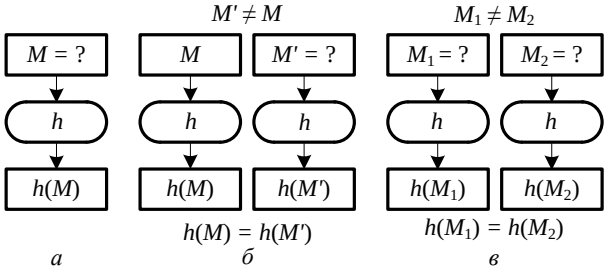

Рисунок 3 – Задачи: *а* – нахождение прообраза; *б* – нахождение второго прообраза; *в* – нахождение коллизии.

[1] Задачи подобного типа имеют универсальный метод решения, вероятность успеха которого всегда равна 1, речь идет о полном переборе всех возможных вариантов. Поэтому когда говорят о вычислительной неразрешимости или трудоемкости ее решения, имеют в виду, что задача не решается за разумное время на современной вычислительной технике.

1.3. Анализ безопасности хеш-функций

При анализе безопасности хеш-функций применяются четыре *проблемы (парадокса) дней рождения*. Дадим два определения.

Случайная величина – это величина, которая принимает в результате опыта одно из нескольких значений, при этом появление какого-либо её значения до измерения этой величины невозможно точно предсказать.

Выборка – это некоторое подмножество значений, которое принимает случайная величина в результате серии опытов.

Сформулируем в качестве примера классический парадокс дней рождения. Задача: сколько учеников нужно собрать в одной классной комнате, чтобы вероятность совпадения даты рождения у двух из них была больше 50%? Парадокс заключается в том, что требуемое число намного меньше, чем количество дней в году.

Пусть k – количество учеников в комнате, $N = 365$ – количество дней в году (для упрощения задачи, високосные годы не учитываются). Решив задачу, найдем

$$k \approx 1{,}18\sqrt{N} \text{ при } P = \frac{1}{2},\ k \approx 23 \text{ при } P = \frac{1}{2} \text{ и } N = 365.$$

Сформулируем все четыре проблемы более формально. Пусть имеется случайная величина, которая с равной вероятностью принимает любое из N возможных значений.

1) Определить минимальное число реализаций k, при котором с вероятностью $P \geq 1/2$ хотя бы одна выборка оказалась равной предопределенной величине.

2) Определить минимальное число реализаций k, при котором с вероятностью $P \geq 1/2$ хотя бы одна выборка оказалась равной выбранной величине.

3) Определить минимальное число реализаций k, при котором с вероятностью $P \geq 1/2$ хотя бы две выборки оказались равными.

Сформируем два набора случайных значений по k выборок в каждом.

4) Определить минимальное число реализаций k, при котором с вероятностью $P \geq 1/2$ хотя бы одна выборка из первого набора оказалась равной одной выборке из второго набора.

Решения проблем дней рождения приведены в таблице 1.

Таблица 1 – Решения проблем дней рождения.

Проблема	Вероятность	Значение k	Значение k при $P = 1/2$	Значение k при $P = 1/2$ и $N = 365$
1	$P \approx 1 - e^{-\frac{k}{N}}$	$k \approx N \ln\frac{1}{1-P}$	$k \approx 0{,}69N$	253
2	$P \approx 1 - e^{-\frac{k-1}{N}}$	$k \approx N \ln\frac{1}{1-P} + 1$	$k \approx 0{,}69N + 1$	254
3	$P \approx 1 - e^{-\frac{k(k-1)}{2N}}$	$k \approx \left(2 \ln\frac{1}{1-P}\right)^{\frac{1}{2}} N^{\frac{1}{2}}$	$k \approx 1{,}18N^{\frac{1}{2}}$	23
4	$P \approx 1 - e^{-\frac{k^2}{2N}}$	$k \approx \left(\ln\frac{1}{1-P}\right)^{\frac{1}{2}} N^{\frac{1}{2}}$	$k \approx 0{,}83N^{\frac{1}{2}}$	16

Классический парадокс дней рождения находится в строке под номером 3.

С помощью проблем дней рождения можно организовать атаку на хеш-функцию. По сути, решение первой проблемы – это решение задачи нахождения первого прообраза (рис. 4,*а*), решение второй проблемы – это нахождение второго прообраза (рис. 4,*б*), решение третьей и четвертой проблемы – это нахождение коллизий.

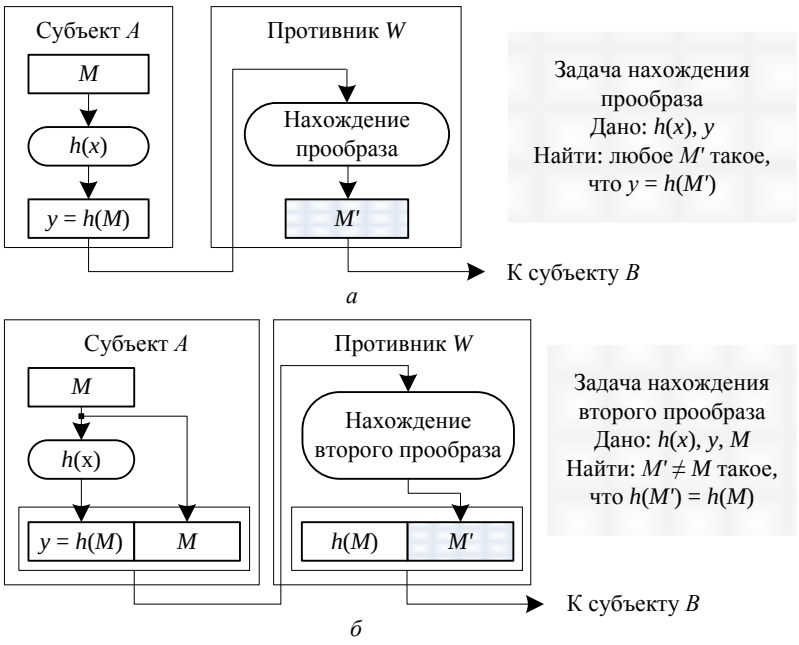

Рисунок 4 – Атаки на хеш-функцию: *а* – нахождение прообраза; *б* – нахождение второго прообраза.

Поиск коллизий в третьем и четвёртом случае отличается. В третьем случае речь идет о поиске двух произвольных сообщений, имеющих одинаковое значение хеш-образа. В четвёртом случае предполагается, что одно сообщение реальное, а другое – фальсифицированное, при этом оба сообщения должны быть осмысленными. Решение в четвертом случае состоит в подготовке второй версии

пересылаемого сообщения (документа), составленной в интересах злоумышленника, и создании двух списков осмысленных сообщений путем внесения избыточности или модификации содержимого (например, добавление пробелов, перестановка слов сообщения, добавление дополнительных избыточных слов и т.п.) без изменения смысла сообщений. Сообщения из первого списка – суть документы, составленные в интересах получателя, аналогичные по смыслу исходному. Сообщения из второго списка – суть документы, полученные путем модификации фальсифицированного сообщения. После этого выполняется сравнение хеш-значений для сообщений из различных списков. Цель – нахождение пары сообщений, для которых эти значения совпадают. После этого возможна, например, компрометация протокола электронной подписи, в котором подпись формируется путем шифрования хеш-образа сообщения на секретном ключе подписывающего.

В таблице 2 приведены оценки сложностей атак на хеш-функцию, где n – разрядность хеш-образа.

Таблица 2 – Оценка сложности атак на хеш-функцию.

Атака	Значение k при $P = 1/2$	Сложность
Нахождение прообраза (1)	$k \approx 0{,}69 \cdot 2^n$	2^n
Нахождение второго прообраза (2)	$k \approx 0{,}69 \cdot 2^n + 1$	2^n
Нахождение коллизии (3)	$k \approx 1{,}18 \cdot 2^{n/2}$	$2^{n/2}$
Нахождение коллизии (4)	$k \approx 0{,}83 \cdot 2^{n/2}$	$2^{n/2}$

При анализе безопасности хеш-функций часто используется модель случайного оракула (Random Oracle, $\mathcal{RO}$) [4, 5]. Случайный оракул – это идеальная криптографический хеш-функция $h(x)$, которая на каждый входной запрос дает случайный ответ, при этом одинаковые запросы будут всегда приводить к одним и тем же ответам случайного оракула независимо от того, когда и как много раз они были сделаны. Не существует формулы или алгоритма вычисления $h(x)$. Есть только один способ узнать $h(x)$ – обратиться к оракулу. $\mathcal{RO}$ – это атомарная или монолитная сущность, которая не может быть разбита на части. Однако на практике хеш-функция не является монолитной, процесс вычисления значения хеш-функции итеративный, при этом на каждой итерации используется примитив следующего уровня, называемый функцией сжатия. Итак, хотя случайного оракула не существует в реальной жизни, можно надеяться, что хорошо спроектированная $h(x)$ будет вести себя подобно $\mathcal{RO}$.

1.3. Итеративная хеш-функция

Наиболее простой и наиболее распространенный способ построения хеш-функции, носит название конструкции Меркля-Дамгарда [1, 2, 4, 6], которая основана на использовании функции сжатия $G : \{0, 1\}^m \times \{0, 1\}^n \rightarrow \{0, 1\}^m$:

$$h_0 = IV;$$
$$h_i = G(h_{i-1}, m_i), i = 1, \dots, t;$$
$$h(M) = h_t,$$

где $M = m_1, \dots, m_t$ – дополненное исходное сообщение, m_i – i-й блок дополненного сообщения, $|h_i| = m, |m_i| = n, IV \in \{0,1\}^m$ – стартовый вектор или вектор инициализации, t – число блоков дополненного сообщения, $h_1, h_2, ..., h_t$ – промежуточные результаты вычисления итераций, число которых равно числу блоков t. Оригинальное сообщение дополняется до длины, кратной n, где n – разрядность блока данных, обрабатываемого функцией сжатия. На i-м итерационном шаге функция сжатия G принимает результат предыдущего шага h_{i-1} и i-й блок данных m_i, а затем формирует результат $h_i = G(h_{i-1}, m_i)$. На шаге t полученное значение h_t объявляется хеш-образом исходного сообщения, т.е. $h(M) = h_t$. Число элементов памяти, необходимых для работы функции G, называется внутренним состоянием.

Слабости конструкции Меркля-Дамгарда демонстрируют так называемые атаки расширения (Extension Attacks). Рассмотрим четыре примера таких атак.

Атака нахождения коллизий. Предположим мы имеем сообщение M длиной $L = |M|$, хеш-образ которого, полученный по схеме Меркля-Дамгарда, равен $h(M)$. Тогда задача нахождения коллизий решается тривиально:

$$h(M \,||\, pad \,||\, x) = h(h(M) \,||\, x),$$

где pad – дополнение, присоединенное к сообщению перед хешированием, и $|\, pad \,| = L \bmod m$, где m – длина блока сообщения M, которая обычно равна $|h(M)|$. Заметим, что $|\, pad \,|$ может быть равно 0, если сообщение выровнено до границы блока. Эта атака становится невозможной при добавлении к сообщению его длины перед хешированием.

Атака нахождения второй коллизии. Задача нахождения новой коллизии решается тривиально путем расширения двух сообщений равной длины, найденных при вычислении первой коллизии. Пусть M и N – два различных сообщения, при этом $h(M) = h(N)$, $|M| = |N|$, тогда вторая коллизия может быть найдена путем расширения M и N с использованием произвольной строки S:

$$h(M \parallel S) = h(N \parallel S).$$

Атака со связанным сообщением (Related Message Attack). Если используется схема Меркля-Дамгарда, можно легко вычислить связанное/расширенное сообщение M' при известном хеш-образе $h(M)$ неизвестного оригинального сообщения M известной длины L. Это $h(M \parallel L \parallel x)$ – хеш-образ сообщения, состоящего из оригинального сообщения M, дополненного строкой $(L \parallel x)$. С другой стороны, так как атакующий знает L, легко вычислить, как M дополняется перед хешированием. Эта атака никак не влияет на стойкость хеш-функции к нахождению коллизий, но показывает, что поведение схемы Меркля-Дамгарда не соответствует модели случайного оракула.

Подделка кода MAC (MAC Forgery Attack). Атака позволяет вычислить корректное сообщение без знания секретного ключа K при использовании алгоритма формирования кода MAC, основанного на схеме Меркля-Дамгарда. Предположим, имеется следующий алгоритм формирования кода аутентификации сообщений: $\mathrm{MAC}(K, M) = h(K \parallel M)$. Тогда можно модифицировать сообщение M путем присоединения строки Y и получить код MAC, соответствующий новому сообщению $K \parallel Y$, без знания ключа K:

$$\mathrm{MAC}(K, M \parallel Y) = h(K \parallel M \parallel Y).$$

В некоторых случаях для нейтрализации слабостей, присущих классической конструкции Меркля-Дамгарда, используется дополнительный финальный шаг преобразования F, в этом случае $h(M) = F(h_t)$. На рис. 5 показана схема такой итерационной хеш-функции, которая является стойкой в смысле нахождения коллизий, если аналогичным свойством обладает используемая функция сжатия G. Кроме того, рекомендуется использовать фиксированный IV, однозначное дополнение и длину сообщения на завершающем этапе преобразования. Типичная структура дополнения показана на рис. 6.

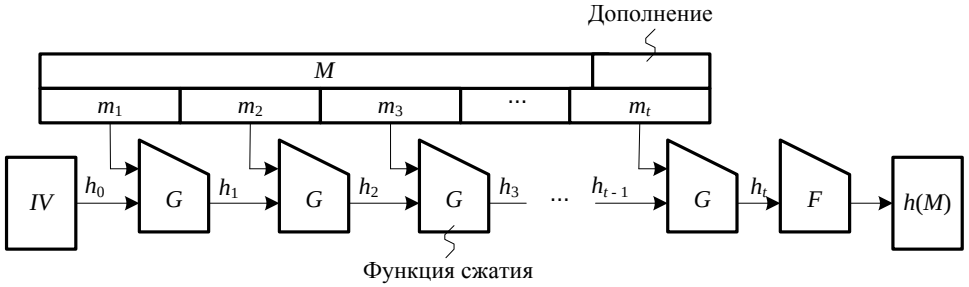

Рисунок 5 – Схема итеративной хеш-функции.

Дополнение	
Padding 1000...000	Длина M

Рисунок 6 – Структура типичного дополнения при итеративном хешировании.

1.4. Хеш-функции на основе итеративных блочных симметричных шифров

Блочный симметричный шифр суть зависящее от секретного ключа $K \in \{0,1\}^k$ преобразование вида E: $\{0, 1\}^m \times \{0, 1\}^k \rightarrow \{0, 1\}^m$, где m – разрядность блока данных, k – разрядность ключа. При использовании для построения $h(x)$ симметричных блочных криптоалгоритмов стойкость хеш-функции гарантируется стойкостью применяемого блочного шифра [4]. Пусть

$$M = m_1 m_2 ... m_i ... m_t, i = 1, ..., t,$$

– последовательность, состоящая из блоков, каждый из которых есть результат расширения блоков исходного сообщения меньшей длины. Наиболее надежные схемы получаются при использовании для вычисления текущего хеш-значения h_i функции шифрования E_K, где в качестве ключа используется предыдущее хеш-значение h_{i-1}; хотя известны схемы, в которых в качестве ключа используется либо очередной блок сообщения m_i, либо $h_{i-1} \oplus m_i$. Наиболее известны следующие схемы:

$$h_i = E_{h_{i-1}}(m_i) \oplus m_i;$$

$$h_i = E_{h_{i-1}}(m_i \oplus h_{i-1}) \oplus m_i \oplus h_{i-1};$$

$$h_i = E_{h_{i-1}}(m_i) \oplus m_i \oplus h_{i-1} \text{ (рис. 7)};$$

$$h_i = E_{h_{i-1}}(m_i \oplus h_{i-1}) \oplus m_i.$$

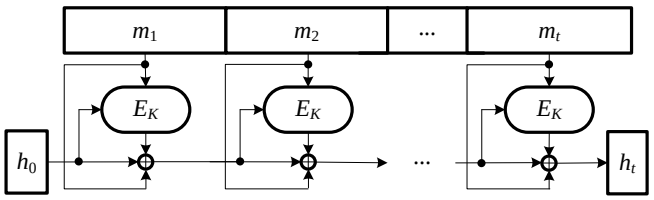

Рисунок 7 – Вариант построения хеш-функции на основе функции шифрования.

1.5. Хеш-функции с архитектурой Квадрат

В 1997 г. НИСТ США объявил о начале программы по принятию нового стандарта криптографической защиты Advanced Encryption Standard (AES), стандарта 21 века для закрытия важной информации правительственного уровня, на замену существующему с 1974 г. алгоритму DES.

В октябре 2000 г. конкурс завершился – победителем был признан бельгийский шифр Rijndael, как имеющий наилучшее сочетание стойкости, производительности, эффективности реализации, гибкости. Его низкие требования к объему памяти делают его идеально подходящим для встроенных систем. Ав-

торами шифра являются J. Daemen и V. Rijmen, начальные буквы фамилий которых и образуют название алгоритма [7].

AES – это итерационный блочный шифр, имеющий длину блоков, равную 128 битам и различные длины ключей. Длина ключа может быть равна 128, 192 или 256 битам.

Промежуточные результаты преобразований, выполняемых в рамках криптоалгоритма, называются *состояниями* (state). Все входные блоки данных, все промежуточные результаты преобразований, все выходные блоки данных, а также ключ шифрования можно представить в виде квадратного массива байтов (рис. 8). Этот массив имеет 4 строки (row) и 4 столбца (columns).

a_{00}	a_{01}	a_{02}	a_{03}
a_{10}	a_{11}	a_{12}	a_{13}
a_{20}	a_{21}	a_{22}	a_{23}
a_{30}	a_{31}	a_{32}	a_{33}

k_{00}	k_{01}	k_{02}	k_{03}
k_{10}	k_{11}	k_{12}	k_{13}
k_{20}	k_{21}	k_{22}	k_{23}
k_{30}	k_{31}	k_{32}	k_{33}

Рисунок 8 – Пример представления состояния и ключа шифрования для AES-128.

В некоторых случаях ключ шифрования рассматривается как линейный массив 4-байтовых слов. Слова состоят из 4 байтов, которые находятся в одном столбце (при представлении в виде прямоугольного массива).

Входные данные для шифра обозначаются как байты состояния в порядке a_{00}, a_{10}, a_{20}, a_{30}, a_{01}, a_{11}, a_{21}, a_{31}, ... После завершения действия шифра выходные данные получаются из байтов состояния в том же порядке.

В состав раунда AES-128 входят следующие 4 преобразования:

– нелинейная замена байтов, выполняемая независимо с каждым байтом состояния, таблица замен *S*-блока размером 8×256 является инвертируемой (SubBytes);
– побайтовый циклический сдвиг строк результата – *i*-я строка сдвигается на *i* байтов влево, $i = \overline{0,3}$ (ShiftRows);
– перемешивание столбцов результата (MixColumns), суть перемешивания – умножение столбца на MDS-матрицу размерностью 4×4;
– поразрядное сложение по модулю 2 (XOR) результата с раундовым ключом (AddRoundKey).

Десятый раунд отличается от остальных – в нем отсутствует предпоследняя операция для более эффективной реализации обратного преобразования.

Раундовые ключи вырабатываются из ключа шифрования посредством *алгоритма выработки ключей* (Key Schedule), который содержит два компонента: *расширение ключа* (Key Expansion) и *выбор раундового ключа* (Round Key Selection). Основополагающие принципы алгоритма выглядят следующим образом:

- общее число бит раундовых ключей равно длине блока, умноженной на число раундов плюс 1 (например, для длины блока 128 бит и 10 раундов требуется 1408 бит ключа);
- ключ шифрования расширяется в *расширенный ключ* (Expanded Key);
- раундовые ключи берутся из расширенного ключа следующим образом: первый раундовый ключ содержит первые 4 слова, второй - следующие 4 слова и т.д.

Расширение ключа (Key Expansion). Расширенный ключ представляет собой линейный массив 4-байтовых слов. Первые 4 слова содержат ключ шифрования. Все остальные слова определяются рекурсивно из слов с меньшими индексами.

Криптоалгоритм. Шифр AES-128 состоит из:

- начального добавления раундового ключа;
- 9 раундов;
- заключительного раунда.

Новая архитектура (рис. 9,*а*), с использованием которой построена функция шифрования E_k, получила название Квадрат. Свойства алгоритма иллюстрирует рис. 9,*б*, из которого видно, что два раунда обеспечивают полное рассеивание и перемешивание информации. Видно, что даже незначительные изменения на входе нелинейной функции приводят к существенным изменениям на выходе (в среднем изменяется половина бит преобразованного блока).

Основные достоинства AES-128:

– новая архитектура Квадрат, обеспечивающая быстрое рассеивание и перемешивание информации, при этом за один раунд преобразованию подвергается весь входной блок;
– байт-ориентированная структура, удобная для реализации на 8-разрядных микроконтроллерах;
– все раундовые преобразования суть операции в конечных полях, допускающие эффективную аппаратную и программную реализацию на различных платформах.

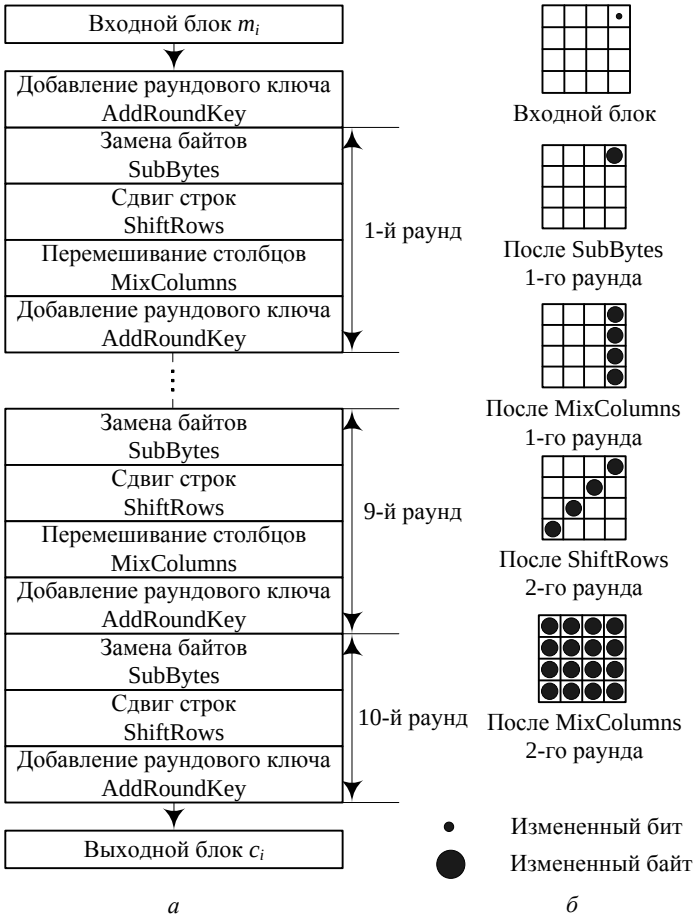

Рисунок 9 – Шифр AES-128: *а* – итерационное блочное преобразование (нелинейная функция зашифрования); *б* – рассеивание и перемешивание информации в процессе двухраундового преобразования.

Возможные направления совершенствования архитектуры Квадрат.

Можно выделить следующие пути совершенствования архитектуры Квадрат:

– замена раундовой операции сдвига строк ShiftRows на операцию перемешивания строк MixRows позволит обеспечить полное рассеивание и перемешивание за один раунд, при этом MDS-матрицы, используемые для перемешивания строк и столбцов, должны быть разными;

– совместное использование архитектур «Петля Фейстеля» и «Квадрат», например использование в качестве раундовой функции *f* операций добавления раундового ключа, замены байтов, перемешивания строк и перемешивания столбцов;
– переход от архитектуры Квадрат к архитектуре Куб.

AES имеет очень простой и даже элегантный дизайн. Архитектура Квадрат открыла новое и очень интересное направление развития симметричных криптоалгоритмов. И практически сразу же после принятия стандарта AES стали появляться хеш-фунции с AES-подобной структурой. Первым таким решением стала хеш-функция Whirlpool [8], представленная в рамках европейского конкурса NESSIE (New European Schemes for Signatures, Integrity and Encryption).

Хеш-функция Whirlpool. При проектировании Whirlpool использовалась структура, показанная на рис. 7. В качестве симметричного блочного шифра для реализации функции сжатия авторами (V. Rijmen, P. Barreto) использован модифицированный алгоритм AES (рис. 10). В таблице 3 приведена сравнительная характеристика блочного шифра W, лежащего в основе Whirlpool , и AES.

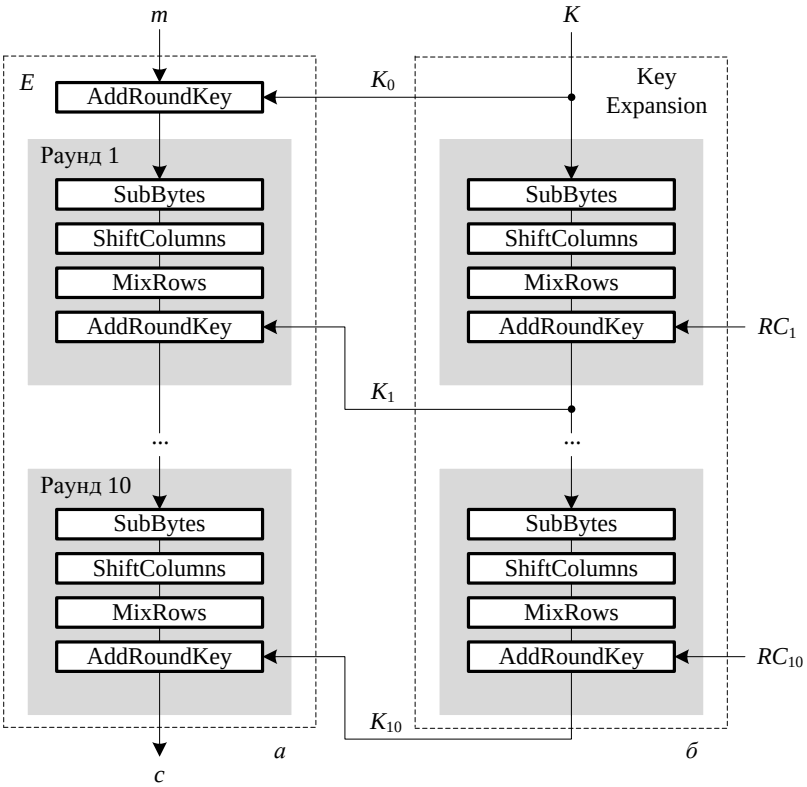

Рисунок 10 – Структура блочного шифра W: *а* – функция шифрования; *б* – процедура «разворачивания» ключа.

Во второй половине 2000-х годов началось массовое появление хеш-функций, использующих идеи, предложенные авторами AES. В качестве примера можно перечислить хеш-функции SHAvite, ECHO, GrØstl и Стрибог [9-12]. Затем стали появляться хеш-функции, использующие многомерные преобразования – CubeHash, Keccak [13]. Можно с полным основанием утверждать, что за подобными алгоритмами будущее.

Таблица 3 – Сравнение блочных шифров W и AES.

	W	AES
Размер блока данных, бит	512	128
Размер ключа, бит	512	128, 192 или 256
Ориентация матрицы	По строкам	По столбцам
Число раундов	10	10, 12 или 14
Процедура разворачивания ключа	Последовательность раундовых функций W	Специальный алгоритм
Многочлен над $GF(2^8)$	$x^8 + x^4 + x^3 + x^2 + 1$ (001D)	$x^8 + x^4 + x^3 + x + 1$ (001C)
S-блок	Рекурсивная структура	Мультипликативная инверсия в $GF(2^8)$ плюс аффинное преобразование
Раундовые константы	Последовательные записи *S*-блока	Элементы 2^i в $GF(2^8)$
Диффузионный слой	MixRows – умножение справа на MDS-матрицу 8 × 8 (1, 1, 4, 1, 8, 5, 2, 9)	MixColumns – умножение слева на MDS-матрицу 4 × 4 (2, 3, 1, 1)
Перестановка	ShiftColumns	ShiftRows

2. Стандарт хеширования SHA-3

2.1. Secure Hash Algorithm (SHA)

Алгоритм SHA является частью стандарта SHS (Secure Hash Standard), разработанного в 1993 г. Национальным институтом стандартов и технологий США (FIPS 180) и Агенством национальной безопасности США. SHA использует принципы, предложенные ранее Р. Ривестом при разработке своих алгоритмов MD4 MD5. В 1995 г. стандарт был пересмотрен (FIPS 180-1) в пользу версии SHA-1. Позднее стандарт был пересмотрен вновь и были определены 4 новые версии: SHA-224, SHA-256, SHA-384, SHA-512. Все версии имеют одинаковую структуру, поэтому часто их называют общим именем SHA-2. В таблице 4 приведены характеристики этих версий.

Таблица 4 – Характеристики SHA.

Характерстики	SHA-1	SHA-224	SHA-256	SHA-384	SHA-512
Максимальная длина сообщения	$2^{64}-1$	$2^{64}-1$	$2^{64}-1$	$2^{128}-1$	$2^{128}-1$
Длина блока	512	512	512	1024	1024
Длина хеш-образа	160	224	256	384	512
Число раундов	80	64	64	80	80
Длина слова	32	32	32	64	64

Многие популярные хеш-функции используют так называемую ARX-конструкцию, предполагающую использование алгебраически несовместимых операций: сложения по модулю 2^n, где $n = 32$ или $n = 64$ (Add); сдвига (Rotate) и

поразрядного сложения по модулю два (XOR). ARX очень сложна для анализа и по этой причине эффективные атаки на нее появляются спустя годы.

Начиная с 2005 года, положение в области криптографических хеш-функций стало напоминать кризисное, и по этой причине в 2007 году НИСТ объявил открытый международный конкурс по выбору нового стандарта хеширования SHA-3. Конкурс был ответом НИСТ на последние достижения в области криптоанализа алгоритмов хеширования.

Конкурс SHA-3 вызвал появление новых подходов к построению хеш-функций:

- конструкции HAIFA, суть которой заключается в двух нововведениях, а именно, в добавлении счетчика итераций и в полной интеграции соли (salt) в структуру хеш-функции [14-16];
- конструкции Wide-pipe, идея которой заключается в увеличении (по сравнению с традиционной конструкцией Narrow-pipe) размера внутреннего состояния [17];
- конструкции Sponge, основанной, как будет показано дальше, на использовании качественного генератора псевдослучайных чисел [18-20].

В конце октября 2008 года НИСТ получил 64 заявки на участие, в конце декабря того же года отобрал среди них 51 кандидата на участие в первом раунде конкурса, в июле 2009 года отобрал 14 кандидатов на участие во втором раунде и, наконец, в декабре 2010 года, – пять участников третьего финального раунда. Финалистами конкурса стали алгоритмы BLAKE (HAIFA), GrØstl (Wide-Pipe), JH (новая конструкция) , Keccak (Sponge) и Skein (Wide-pipe).

На всех этапах конкурса в его работе активно участвовало мировое криптографическое сообщество. Основываясь на открытых комментариях, результатах криптоанализа, особенностей аппаратной и программной реализации, в октябре 2012 года НИСТ объявил алгоритм Keccak победителем, завершив пятилетний конкурс[2].

2.2. Конструкция Sponge

Итак, в качестве основы для нового стандарта SHA-3 была выбрана хеш-функция Keccak, построенная с использованием конструкции под названием Sponge (губка), которая показана на рис. 11.

[2] Хотя конкурс завершился в конце 2012 года, сам новый стандарт SHA-3 до сих пор не опубликован.

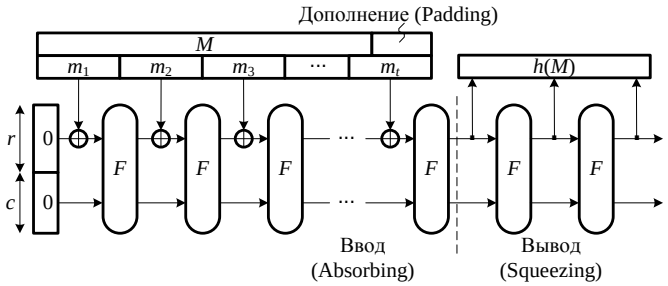

Рисунок 11 – Схема Sponge (в задаче хеширования данных).

Схема Sponge имеет внутреннее состояние S разрядностью b, $b = r + c$, значение r называется *rate* (определяет скорость преобразования), а значение c (параметр безопасности) – *capacity*. Иначе говоря, состояние делится на две части: S_1 разрядностью r и S_2 разрядностью c. Пусть RL(Resistance Level) – уровень стойкости, тогда по утверждению авторов конструкции справедливо соотношение $c \geq 2RL$ (иначе говоря, Sponge гарантирует нижнюю границу сложности $2^{c/2}$ для любой атаки на хеш-функцию).

Процесс хеширования состоит из двух многораундовых этапов: ввод (Absorbing, впитывание) и вывод (Squeezing, выжимание).

Этап ввода информации:

- Состояние S инициализируется нулями. Сообщение M дополняется до длины кратной r, а затем разбивается на блоки длины r;
- Первый блок сообщения M суммируется по модулю 2 (операция XOR) с S_1; результат операции XOR и S_2 передаются на вход раундовой функции F;
- Второй блок сообщения M складывается по модулю 2 с первыми r битами выхода функции F; результат операции XOR и последние c бит результата действия функции f вновь передаются на вход функции F второго раунда;
- Этап продолжается до тех пор, пока не будут обработаны все блоки сообщения M.

Особенностью этапа является в том, что на каждом раунде блок сообщения складывается по модулю 2 только с частью состояния, а функция F производит преобразование всего состояния и делает его зависимым от всего сообщения M.

Этап вывода информации.

Результат длины b, полученный на этапе впитывания, вновь подаётся на вход функции F . При этом с выхода этой функции считываются первые r бит. Этот процесс повторяется конечное число раз, при этом над r битами, считанными на каждом этапе, производится операция конкатенации (сцепления блоков).

Повторение происходит до тех пор, пока не получим результат, т.е. хеш-образ, нужной длины.

Важно отметить, что функциональные возможности схемы Sponge значительно шире, чем только хеширование данных. Рассмотрим другие варианты применения схемы Sponge.

MAC (Message Authentication Code, код аутентификации сообщения) – код, добавляемый к сообщению для обеспечения его целостности (аутентичности). Для формирования этого кода достаточно на вход схемы добавить секретный ключ (Key). Схема формирования MAC показана на рис. 12,*а*.

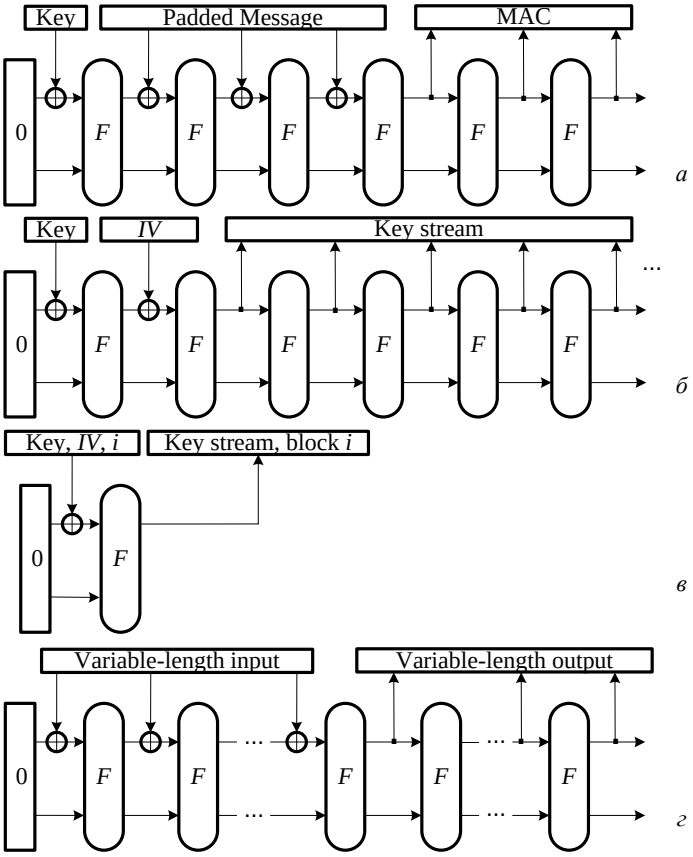

Рисунок 12 – Варианты использования схемы SPONGE: *а* – формирование кода MAC; *б* – генерация PRS при синхронном поточном шифровании; *в* – PRNG с произвольным доступом; *г* – преобразование входной последовательности произвольной длины в выходную последовательность произвольной длины

Добавив секретный ключ (Key) с открытым вектором инициализации (*IV*) и вывод гаммы (Key Stream) произвольной длины, можно получить поточный шифр, схема построения которого представлена на рис. 12,*б*. На рис. 12,*в* показана схема PRNG с произвольным доступом к отдельным элементам выходной последовательности. Использование входа и выхода переменной длины в схеме SPONGE (рис. 12,*г*) может применяться, например, для генерации симметричных ключей из паролей.

Наконец, возможно хеширование с солью (Salt) и при необходимости с замедлением (рис. 13), что целесообразно при организации парольных систем разграничения доступа и, в частности, для противодействия атакам с использованием радужных таблиц (Rainbow Tables).

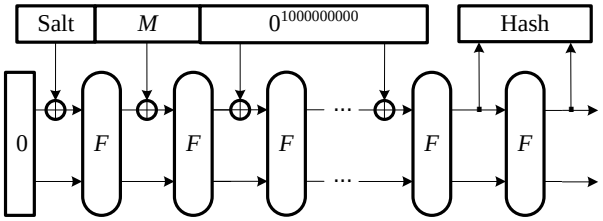

Рисунок 13 – Схема хеширования с солью и искусственным замедлением.

Другие варианты использования Sponge рассмотрены в [18-20].

Схему Sponge можно изобразить по-другому, а именно так, как показано на рис. 14. После этого становится понятно, что схема не новая, так как практически не отличается от схемы, показанной ранее на рис. 1. По такому же принципу построена, например, схема формирования контрольного кода CRC. Иначе говоря, заслугой авторов Sponge является то, что они раскрыли многочисленные возможности этой схемы с точки зрения построения различных примитивов симметричной криптографии.

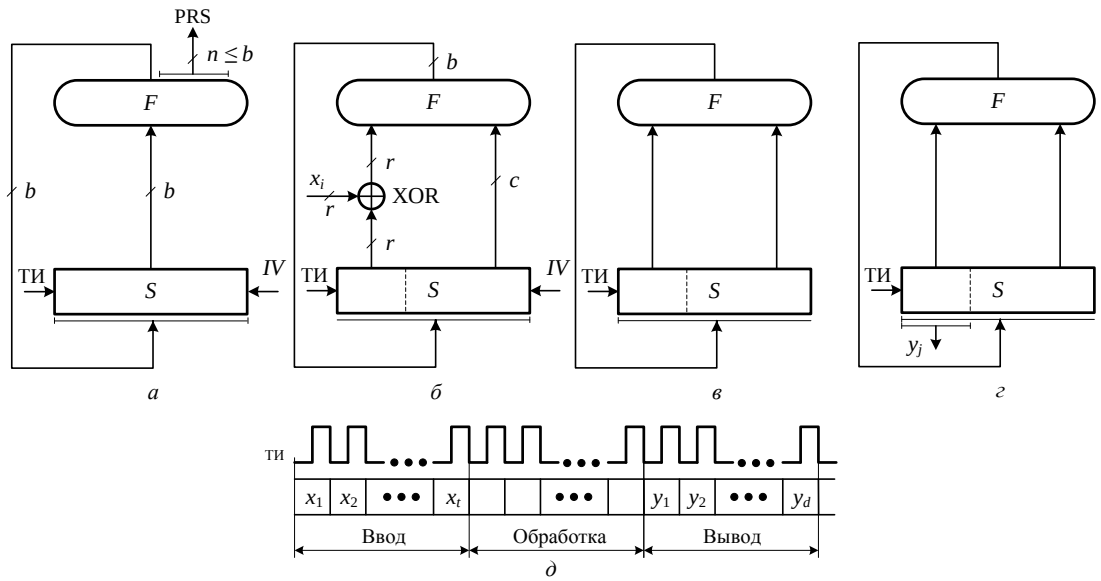

Рисунок 14 – Схема Sponge (другой взгляд); *а* – PRNG; *б* - схема преобразования в фазе ввода; *в* – схема преобразования в фазе обработки; *г* – схема преобразования в фазе вывода информации; *в* – временная диаграмма ввода, обработки входной и вывода выходной последовательностей.
PRS – псевдослучайная последовательность, ТИ – тактовые импульсы, x_i, y_i – соответственно элементы входной и выходной последовательностей, $i = 1, \ldots, t$; $j = 1, \ldots, d$.

2.2. Алгоритм Keccak

Хеш-функция Keccak [13] определяет целое семейство функций Keccak-$F[b]$ с разрядностью состояния

$$b \in \{25, 50, 100, 200, 400, 800, 1600\}.$$

Однако, хотя пользователь в принципе может выбирать для своей реализации любую из предложенных авторами функций, следует иметь в виду, что в качестве стандарта SHA-3 принята только функция Keccak-1600. В этом случае авторы рекомендуют использовать следующие параметры:

SHA-224: r = 1152, c = 448 (вернуть первые 28 байт результата);

SHA-256: r = 1088, c = 512 (вернуть первые 32 байт результата);

SHA-384: r = 832, c = 768 (вернуть первые 48 байт результата);

SHA-512: r = 576, c = 1024 (вернуть первые 64 байт результата).

Последний вариант будет использован в стандарте.

Внутреннее состояние. Алгоритм работает со словами длиной $w = 2^l$, $0 < l \leq 6$. Разрядность состояния равна $b = 5 \times 5 \times 2^l$ бит. Состояние (state) рассматривается как трехмерный массив $S[x][y][z]$, в составе которого выделяются столбцы (columns), строки (rows), линии (lanes) длиной 2^l бит и срезы (slices), как показано на рис. 15. Срез представляет собой квадратный массив 5×5 бит.

Длина слова w определяет соответствующую функцию Keccak-$F[b]$. В SHA-3 используется функция Keccak-F[1600].

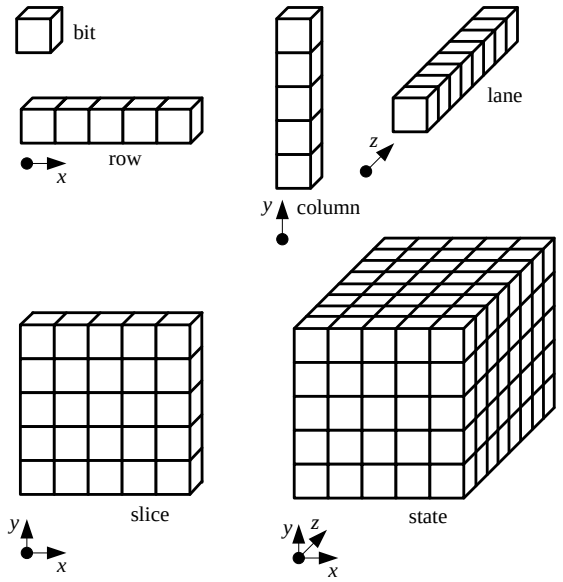

Рисунок 15 – Состояние Keccak (на примере b = 200).

Инициализация и дополнение.

Вектор инициализации IV для конструкции Sponge:

$$S[x][y][z] \leftarrow 0 \text{ for all } 0 \le x, y \le 4,\ 0 \le z \le 2^l - 1.$$

Дополнение последнего блока:

- Добавляется единичный бит;
- Добавляется минимальное число нулевых бит для достижения промежуточной длины последнего блока (r - 1);
- Добавляется финальный единичный бит.

Раундовая функция.

Число раундов равно 12 + 2l. В SHA-3 для 64-разрядных процессоров l = 6, поэтому число раундов равно 24. Каждый раунд суть композиция 5 раундовых преобразований

$$S \leftarrow \iota \circ \chi \circ \pi \circ \rho \circ \theta(S).$$

Каждое преобразование оперирует либо только строками, либо только столбцами, либо только линиями, либо является перестановкой на плоскости (x, y).

Формальное описание theta-функции θ.

Суть преобразования – замена каждого бита суммой по модулю 2 данного бита и битов четности ($parity_1$ и $parity_2$) соседних столбцов:

$$parity_1 \leftarrow \sum_{\mu=0}^{4} S[x-1][\mu][z] \bmod 2,$$

$$parity_2 \leftarrow \sum_{\mu=0}^{4} S[x+1][\mu][z] \bmod 2,$$

$$S[x][y][z] \leftarrow S[x][y][z] \oplus parity_1 \oplus parity_2.$$

Строки и столбцы индексируются в GF(5), линии – по модулю 2^l, т.е. равному длине слова.

Формальное описание rho-функции ρ.

Суть преобразования – сдвиг бит линии (lane), при этом величина сдвига равна $\frac{(t+1)(t+2)}{2} \bmod 2^l$ для $-1 \le t \le 23$ (таблица 5):

1) Вычислить (все операции осуществляются в $GF(5)$) для $0 \le t \le 23$ координаты строки и столбца

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix}^t \cdot \begin{pmatrix} 1 \\ 0 \end{pmatrix};$$

2) Дополнительно установить $t = -1$ для $x = y = 0$;

$$\rho: S[x][y][z] \leftarrow S[x][y][z - \frac{(t+1)(t-1)}{2}].$$

Шаблон вычисления индексов:

1) $t = 0$, сдвиг осуществляется на $\frac{(t+1)(t+2)}{2} = 1$:

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix}^0 \cdot \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix};$$

2) $t = 1$, сдвиг осуществляется на $\frac{(t+1)(t+2)}{2} = 3$:

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix}^1 \cdot \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 2 \end{pmatrix};$$

3) $t = 2$, сдвиг осуществляется на $\frac{(t+1)(t+2)}{2} = 6$:

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix}^2 \cdot \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 2 & 3 \\ 6 & 11 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 2 \\ 1 \end{pmatrix}.$$

Таблица 5 – Величины сдвига линий для всех возможных значений $\begin{pmatrix} x \\ y \end{pmatrix}$.

	$x = -2$	$x = -1$	$x = 0$	$x = 1$	$x = 2$
$y = 2$	25	39	3	10	43
$y = 1$	55	20	36	44	6
$y = 0$	28	27	0	1	62
$y = -1$	56	14	18	2	61
$y = -2$	21	8	41	45	15

Формальное описание pi-функции π (рис. 16).

Суть преобразования – перестановка линий, иначе говоря, перестановка в плоскости (x, y):

$$\begin{pmatrix} x \\ y \end{pmatrix} \leftarrow \begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} y \\ 2x + 3y \end{pmatrix}.$$

Все вычисления осуществляются по модулю 5. Следует иметь в виду, что на рис. 16 точка (0, 0) находится в центре.

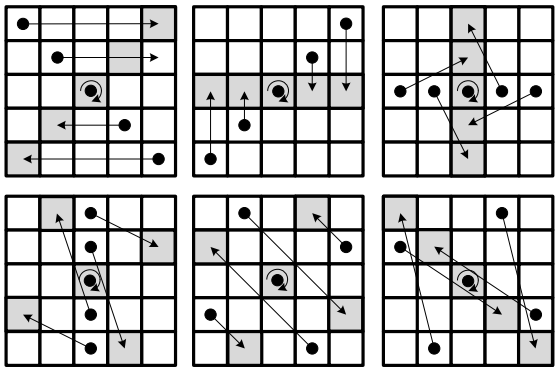

Рисунок 16 – Преобразование π.

Таким образом, получаем:

$$\pi : S[x][y][z] \leftarrow S[y][2x + 3y][z].$$

Примеры:

– Верхний левый угол $\begin{pmatrix}-2\\2\end{pmatrix}\mapsto\begin{pmatrix}2\\2\cdot(-2)+3\cdot 2\end{pmatrix}=\begin{pmatrix}y\\2x+3y\end{pmatrix}=\begin{pmatrix}2\\2\end{pmatrix}$;

– $\begin{pmatrix}2\\1\end{pmatrix}\mapsto\begin{pmatrix}1\\2\cdot 2+3\cdot 1\end{pmatrix}=\begin{pmatrix}1\\7\end{pmatrix}=\begin{pmatrix}1\\2\end{pmatrix}$.

Формальное описание chi-функции χ (рис. 17).

Суть преобразования – замена бита путем комбинирования его с двумя последующими битами строки, индексы *x* и *y* при этом не меняются:

$$\chi: S[x][y][z] \leftarrow S[x][y][z] \oplus (\neg S[x+1][y][z] \wedge S[x+2][y][z]).$$

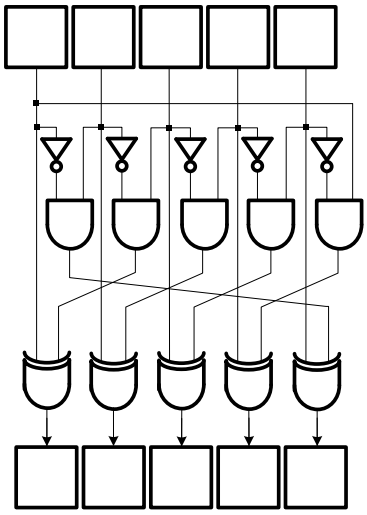

Рисунок 17 – Преобразование χ.

Формальное описание iota-функции ι.

Суть преобразования – сложение (XOR) линии $S[x][y][.]$ с раундовой константой:

$$\iota: S[0][0][.] \leftarrow S[0][0][.] \oplus \mathrm{RC}[\mu], 0 \le \mu \le 12 + 2l = 24.$$

RC[0] = 0x0000000000000001,
RC[1] = 0x0000000000008082,
RC[2] = 0x800000000000808A,
RC[3] = 0x8000000000008000,
RC[4] = 0x000000000000808B,
RC[5] = 0x0000000080000001,
RC[6] = 0x8000000080008081,
RC[7] = 0x8000000000008009,
RC[8] = 0x000000000000008A,
RC[9] = 0x0000000000000088,
RC[10] = 0x0000000080008009,
RC[11] = 0x000000008000000A,
RC[12] = 0x000000008000808B,
RC[13] = 0x800000000000008B,
RC[14] = 0x8000000000008089,
RC[15] = 0x8000000000008003,
RC[16] = 0x8000000000008002,
RC[17] = 0x8000000000000080,
RC[18] = 0x000000000000800A,
RC[19] = 0x800000008000000A,
RC[20] = 0x8000000080008081,
RC[21] = 0x8000000000008080.
RC[22] = 0x0000000080000001,
RC[23] = 0x8000000080008008.

Ниже приведен псевдокод алгоритма, при этом, как справедливо отмечается авторами, реально алгоритм гораздо проще, чем кажется на первый взгляд и существует много методов его эффективной реализации [20, 21].

```
---------------------------------------------------------------------
KECCAK-F[b](A)

     For all i  in 0 … (n_r  - 1)
               A = Round[b](A, RC[i])
     return A
---------------------------------------------------------------------
Round[b](A, RC)

     // θ step
C[x] = A[x, 0] ⊕ A[x, 1] ⊕ A[x, 2] ⊕ A[x, 3] ⊕ A[x, 4],   for all  x in 0 … 4
D[x] = C[x - 1] ⊕ ROT(C[x + 1], 1),                       for all  x in 0 … 4
A[x, y] = A[x, y] ⊕ D[x],                          for all  (x, y) in (0 … 4, 0 … 4)

     // ρ and π steps
B[y, 2x + 3y] = ROT(A[x, y], r[x, y]),             for all  (x, y) in (0 … 4, 0 … 4)

     // χ step
A[x, y] = B[x, y] ⊕ ((NOT B[x + 1, y]) AND B[x + 2, y]),
                                                   for all  (x, y) in (0 … 4, 0 … 4)

     // ι step
A[0, 0] = A[0, 0] ⊕ RC

     return A
-------------------------------------------------------------------------------------
Keccak[r, c](M)

     // Padding
     P = M || 0x01 || 0x00 || . . . || 0x00
     P = P ⊕ 0x00 || . . . || 0x00 || 0x80

     // Initialization
     S[x, y] = 0,                                  for all  (x, y) in (0 … 4, 0 … 4)

     // Absorbing phase
     for every block P_i in P
          S[x, y] = S[x, y] ⊕ P_i[x + 5y],      for all (x, y) such that x + 5y < r/w
          S = Keccak-F[r + c](S)

     // Squeezing phase
     Z = empty string
     while output is requested
          Z = Z || S[x, y],                     for all (x, y) such that x + 5y < r/w
          S = Keccak-F[r + c](S)

     Return Z
-------------------------------------------------------------------------------------
```

Примечание. ROT(*w*, *r*) – операция циклического сдвига, которая перемещает бит слова *w* из позиции *i* в позицию *i* + *r* по модулю, равному длине слова (lane).

3. Новый российский стандарт хеширования

В конце 2012 года как ответ на принятие стандарта SHA-3 был принят новый российский стандарт хеширования ГОСТ 34.11 – 2012 [9]. Новый алгоритм хеширования получил название Стрибог. Размер блоков сообщения и внутреннего состояния хеш-функции составляет 512 бит. Стандарт определяет две функции хеширования – «Стрибог-256» и «Стрибог-512» с длинами хеш-образа 256 и 512

бит соответственно. Эти функции отличаются начальным внутренним состоянием и его частью, принимаемой за результат вычислений.

Хеш-функция построена с использованием архитектуры Квадрат, так как внутреннее состояние функции сжатия можно представить в виде квадратного массива байтов 8 × 8. Хеш-функция построена на основе блочного шифра, в котором произведены некоторые принципиальные изменения по сравнению с AES.

В функции сжатия используются только преобразование *LPSX*. Преобразование *LPSX* суть композиция четырех преобразований *S*, *P*, *L* и *X*, описанных ниже. Вместе со сложением по модулю 2^{512} они составляют полный набор операций, использующихся в стандарте.

3.1. Вектора инициализации

Значение вектора IV для хеш-функции «Стрибог-512» равно 0^{512}. Значение вектора IV для хеш-функции «Стрибог-256» равно $(00000001)^{64}$.

3.2. Итерационные константы

Итерационные константы записаны ниже в шестнадцатеричном виде.

C_1 = b1085bda1ecadae9ebcb2f81c0657c1f2f6a76432e45d016714eb88d7585c4fc
4b7ce09192676901a2422a08a460d31505767436cc744d23dd806559f2a64507;
C_2 = 6fa3b58aa99d2f1a4fe39d460f70b5d7f3feea720a232b9861d55e0f16b50131
9ab5176b12d699585cb561c2db0aa7ca55dda21bd7cbcd56e679047021b19bb7;
C_3 = f574dcac2bce2fc70a39fc286a3d843506f15e5f529c1f8bf2ea7514b1297b7b
d3e20fe490359eb1c1c93a376062db09c2b6f443867adb31991e96f50aba0ab2;
C_4 = ef1fdfb3e81566d2f948e1a05d71e4dd488e857e335c3c7d9d721cad685e353f
a9d72c82ed03d675d8b71333935203be3453eaa193e837f1220cbebc84e3d12e;
C_5 = 4bea6bacad4747999a3f410c6ca923637f151c1f1686104a359e35d7800fffbd
bfcd1747253af5a3dfff00b723271a167a56a27ea9ea63f5601758fd7c6cfe57;
C_6 = ae4faeae1d3ad3d96fa4c33b7a3039c02d66c4f95142a46c187f9ab49af08ec6
cffaa6b71c9ab7b40af21f66c2bec6b6bf71c57236904f35fa68407a46647d6e;
C_7 = f4c70e16eeaac5ec51ac86febf240954399ec6c7e6bf87c9d3473e33197a93c9
0992abc52d822c3706476983284a05043517454ca23c4af38886564d3a14d493;
C_8 = 9b1f5b424d93c9a703e7aa020c6e41414eb7f8719c36de1e89b4443b4ddbc49a
f4892bcb929b069069d18d2bd1a5c42f36acc2355951a8d9a47f0dd4bf02e71e;
C_9 = 378f5a541631229b944c9ad8ec165fde3a7d3a1b258942243cd955b7e00d0984
800a440bdbb2ceb17b2b8a9aa6079c540e38dc92cb1f2a607261445183235adb;
C_{10} = abbedea680056f52382ae548b2e4f3f38941e71cff8a78db1fffe18a1b336103
9fe76702af69334b7a1e6c303b7652f43698fad1153bb6c374b4c7fb98459ced;
C_{11} = 7bcd9ed0efc889fb3002c6cd635afe94d8fa6bbbebab07612001802114846679
8a1d71efea48b9caefbacd1d7d476e98dea2594ac06fd85d6bcaa4cd81f32d1b;
C_{12} = 378ee767f11631bad21380b00449b17acda43c32bcdf1d77f82012d430219f9b
5d80ef9d1891cc86e71da4aa88e12852faf417d5d9b21b9948bc924af11bd720.

3.3. Используемые преобразования

X-преобразование. Суть преобразования – поразрядное сложение по модулю два (XOR) двух последовательностей длиной 512 бит каждая (иначе говоря, преобразование аналогично AddRoundKey):

$$X[k] : \{0,1\}^{512} \to \{0,1\}^{512}, X[k](a) = k \oplus a; k, a \in \{0,1\}^{512}.$$

S-преобразование. Суть преобразования – замена байтов, выполняемая независимо с каждым байтом состояния в соответствии с фиксированной таблицей подстановок размером 8×256, каждый байт входной последовательности заменяется соответствующим байтом из таблицы подстановок (аналогично SubBytes):

$$S: \{0,1\}^{512} \to \{0,1\}^{512}, S(a) = S(a_{63}\|\ldots\|a_0) = \pi(a_{63})\|\ldots\|\pi(a_0),$$

где нелинейное биективное преобразование байтов состояния задается подстановкой $\pi: \{0,1\}^8 \to \{0,1\}^8$.

Значения подстановки π записаны ниже в виде массива

$$\pi = (\pi(0), \pi(1), \ldots, \pi(255)):$$

π = (252, 238, 221, 17, 207, 110, 49, 22, 251, 196, 250, 218, 35, 197, 4, 77, 233, 119, 240, 219, 147, 46, 153, 186, 23, 54, 241, 187, 20, 205, 95, 193, 249, 24, 101, 90, 226, 92, 239, 33, 129, 28, 60, 66, 139, 1, 142, 79, 5, 132, 2, 174, 227, 106, 143, 160, 6, 11, 237, 152, 127, 212, 211, 31, 235, 52, 44, 81, 234, 200, 72, 171, 242, 42, 104, 162, 253, 58, 206, 204, 181, 112, 14, 86, 8, 12, 118, 18, 191, 114, 19, 71, 156, 183, 93, 135, 21, 161, 150, 41, 16, 123, 154, 199, 243, 145, 120, 111, 157, 158, 178, 177, 50, 117, 25, 61, 255, 53, 138, 126, 109, 84, 198, 128, 195, 189, 13, 87, 223, 245, 36, 169, 62, 168, 67, 201, 215, 121, 214, 246, 124, 34, 185, 3, 224, 15, 236, 222, 122, 148, 176, 188, 220, 232, 40, 80, 78, 51, 10, 74, 167, 151, 96, 115, 30, 0, 98, 68, 26, 184, 56, 130, 100, 159, 38, 65, 173, 69, 70, 146, 39, 94, 85, 47, 140, 163, 165, 125, 105, 213, 149, 59, 7, 88, 179, 64, 134, 172, 29, 247, 48, 55, 107, 228, 136, 217, 231, 137, 225, 27, 131, 73, 76, 63, 248, 254, 141, 83, 170, 144, 202, 216, 133, 97, 32, 113, 103, 164, 45, 43, 9, 91, 203, 155, 37, 208, 190, 229, 108, 82, 89, 166, 116, 210, 230, 244, 180, 192, 209, 102, 175, 194, 57, 75, 99, 182).

P-преобразование. Суть преобразования – транспонирование матрицы состояний, иначе говоря, перестановка – для каждой пары байт из входной последовательности выполняется замена одного байта другим:

$$P: \{0,1\}^{512} \to \{0,1\}^{512}, P(a) = P(a_{63}\|\ldots\|a_0) = a_{\tau(63)}\|\ldots\|a_{\tau(0)}.$$

Перестановка $\tau \in S_{64}$ является фиксированной и в виде массива $\tau(0), \tau(1), \ldots, \tau(63)$ выглядит следующим образом.

τ = (0, 8, 16, 24, 32, 40, 48, 56, 1, 9, 17, 25, 33, 41, 49, 57, 2, 10, 18, 26, 34, 42, 50, 58, 3, 11, 19, 27, 35, 43, 51, 59, 4, 12, 20, 28, 36, 44, 52, 60, 5, 13, 21, 29, 37, 45, 53, 61, 6, 14, 22, 30, 38, 46, 54, 62, 7, 15, 23, 31, 39, 47, 55, 63).

L-преобразование. Линейное преобразование векторов (строк матрицы состояния) задается умножением справа на матрицу A над $GF(2)$ размером 64×64 (аналогично MixRows):

$$L: \{0,1\}^{512} \to \{0,1\}^{512}, L(a) = L(a_7\|\ldots\|a_0) = l(a_7)\|\ldots\|l(a_0).$$

Строки матрицы A записаны ниже в шестнадцатеричном виде, при этом в одной строке записано 4 строки матрицы, при этом в строке с номером i, $i = 0, \ldots, 15$ записаны строки матрицы A с номерами $4i + j$, $j = 0, \ldots, 3$.

8e20faa72ba0b470	47107ddd9b505a38	ad08b0e0c3282d1c	d8045870ef14980e
6c022c38f90a4c07	3601161cf205268d	1b8e0b0e798c13c8	83478b07b2468764
a011d380818e8f40	5086e740ce47c920	2843fd2067adea10	14aff010bdd87508
0ad97808d06cb404	05e23c0468365a02	8c711e02341b2d01	46b60f011a83988e
90dab52a387ae76f	486dd4151c3dfdb9	24b86a840e90f0d2	125c354207487869
092e94218d243cba	8a174a9ec8121e5d	4585254f64090fa0	accc9ca9328a8950
9d4df05d5f661451	c0a878a0a1330aa6	60543c50de970553	302a1e286fc58ca7
18150f14b9ec46dd	0c84890ad27623e0	0642ca05693b9f70	0321658cba93c138
86275df09ce8aaa8	439da0784e745554	afc0503c273aa42a	d960281e9d1d5215
e230140fc0802984	71180a8960409a42	b60c05ca30204d21	5b068c651810a89e
456c34887a3805b9	ac361a443d1c8cd2	561b0d22900e4669	2b838811480723ba
9bcf4486248d9f5d	c3e9224312c8c1a0	effa11af0964ee50	f97d86d98a327728
e4fa2054a80b329c	727d102a548b194e	39b008152acb8227	9258048415eb419d
492c024284fbaec0	aa16012142f35760	550b8e9e21f7a530	a48b474f9ef5dc18
70a6a56e2440598e	3853dc371220a247	1ca76e95091051ad	0edd37c48a08a6d8
07e095624504536c	8d70c431ac02a736	c83862965601dd1b	641c314b2b8ee083

3.4.Функция сжатия

На каждой итерации используется функция сжатия

$$G_N : \{0,1\}^{512} \times \{0,1\}^{512} \to \{0,1\}^{512}, N \in \{0,1\}^{512},$$

значение которой вычисляется по формуле

$$G_N(h,m) = E(LPS(h \oplus N), m) \oplus h \oplus m,$$

где

$$E(k,m) = X[K_{13}] \circ LPSX[K_{12}] \circ \ \dots \ \circ LPSX[K_2] \circ LPSX[K_1](m).$$

Значения $K_i \in \{0,1\}^{512}, i = 1, \dots, 13$ вычисляются следующим образом

$$K_1 = K;$$

$$K_i = LPS(K_{i-1} \oplus C_{i-1}), i = 2, \dots, 13.$$

3.5. Вычисление значения хеш-функции

Схема хеширования показана на рис. 18, схема функции сжатия – на рис. 19.

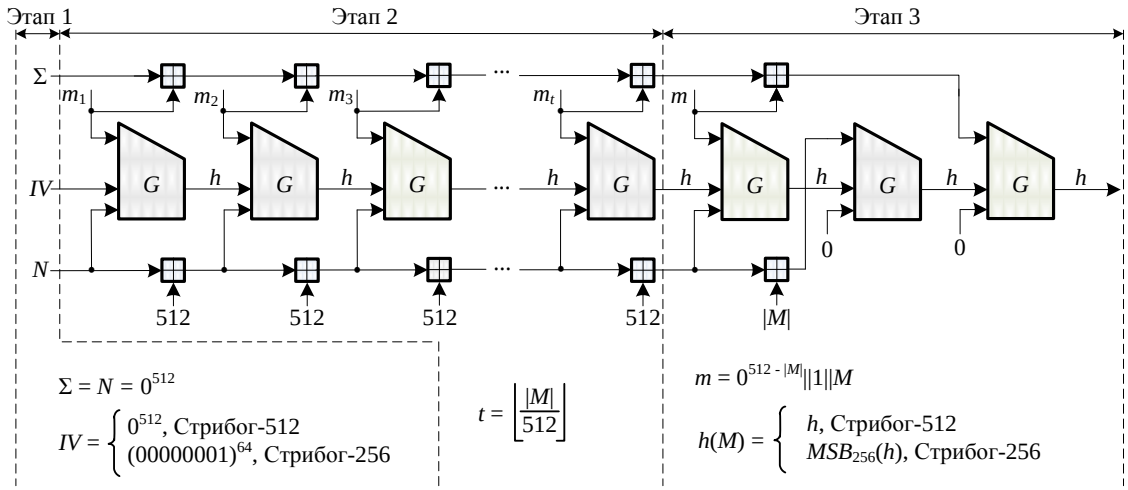

Рисунок 18 – Стрибог: процесс вычисления хеш-функции.

Этап 1.

1.1. Для хеш-функции с длиной выхода 512 бит: $h = IV = 0x00^{64}$. Для хеш-функции с длиной выхода 256 бит: $h = IV = 0x01^{64}$.

1.2. $N = 0^{512}$.

1.3. $\Sigma = 0^{512}$.

Этап 2.

2.1. Проверить условие $|M| < 512$. Если условие выполняется перейти к этапу 3.

2.2. Выделить очередной блок сообщения m, где $m \in \{0,1\}^{512}, M = M' \| m$.

2.3. $h = G_N(m, h)$.

2.4. $N = (N + 512) \bmod 2^{512}$.

2.5. $\Sigma = (\Sigma + m) \bmod 2^{512}$.

2.6. $M = M'$.

2.7. Перейти к шагу 2.1.

Этап 3.

3.1. Произвести дополнение сообщения M до длины в 512 бит по следующему правилу: $m = 0^{511 - |M|} \| 1 \| M$, где $|M|$ - длина сообщения M в битах.

3.2. $h = G_N(m, h)$.

3.3. $N = (N + |M|) \bmod 2^{512}$.

3.4. $\Sigma = (\Sigma + m) \bmod 2^{512}$.

3.5. $h = G_0(h, N)$.

3.6. $h = G_0(h, \Sigma)$.

3.7. Для хеш-функции с длиной выхода в 512 бит возвращаем h в качестве результата. Для функции с длиной выхода 256 бит возвращаем $MSB_{256}(h)$, где отображение $MSB_n : \{0,1\}^* \to \{0,1\}^n$ ставит в соответствие строке $z_{k-1} \dots z_1 z_0, k \geq n$, строку $z_{k-1} \dots z_{k-n+1} z_{k-n}, z_i \in \{0,1\}, i = 0, \dots, (k-1)$.

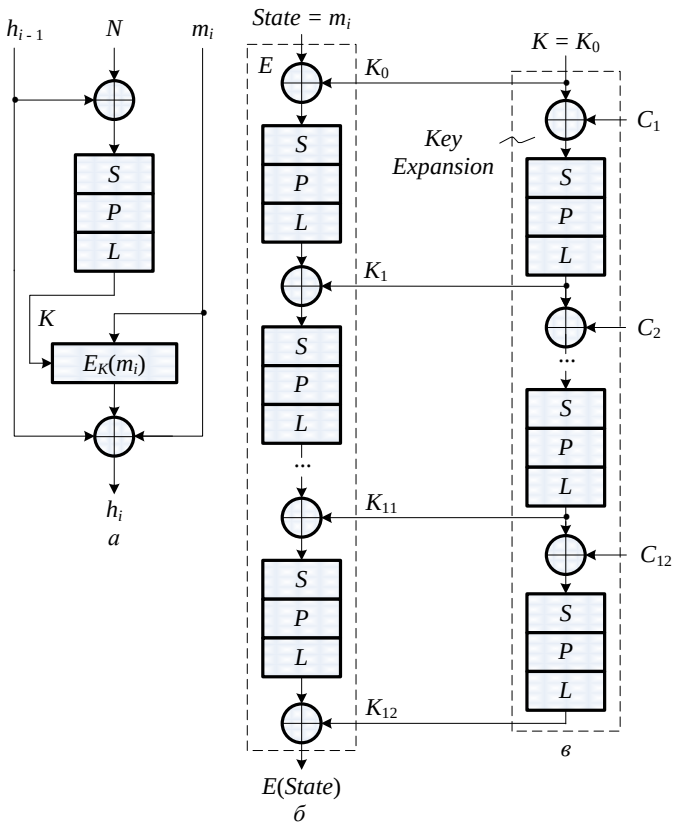

Рисунок 19 – Стрибог: *а* – схема функции сжатия; *б* и *в* – схема блочного шифра, лежащего в основе алгоритма (соответственно структура функции шифрования *E* и последовательность «разворачивания» ключа).

Заключение

В статье кратко рассмотрены основы теории криптографических хеш-функций, приведены примеры их использования в задачах защиты информации. Отмечено, что задача построения качественной хеш-функции сложнее, чем задача построения симметричного блочного шифра.

Отмечена тенденция последних лет, а именно массовое появление хеш-функций, использующих многомерные преобразования.

Описаны особенности нового подхода к построению хеш-функций, основанного на использовании конструкции Sponge. Показано, что, не смотря на то, что сама схема давно известна, удалось найти множество оригинальных режимов ее использования, в том числе в качестве различных примитивов симметричной криптографии. Также показано, что в основе Sponge лежит генератор псевдослучайных чисел (PRNG). Это означает, что при исследованиях статистических свойств хеш-функций могут использоваться многочисленные статистические тесты, разработанные для оценки качества PRNG [23, 24].

Рассмотрены два стандарта на алгоритмы хеширования, а именно, использующий трехмерные преобразования алгоритм Keccak, победивший в конкурсе на приятие нового стандарта SHA-3, и использующий двухмерные преобразования

алгоритм Стрибог, принятый в качестве нового российского стандарта ГОСТ 34.11-2013.

Алгоритм Стрибог основан на использовании блочного шифра с архитектурой Квадрат, впервые предложенной авторами криптоалгоритмов Square и Rijndael, при этом содержит несколько принципиальных отличий от известных решений, в частности можно отметить реализацию преобразования перемешивания строк (L) в виде умножения на матрицу над $GF(2)$ и ввод в состав раунда операции транспонирования матрицы состояния, что фактически обеспечивает при выполнении многораундовых преобразований чередование преобразований перемешивания строк и столбцов матрицы состояния, что очевидно приводит к более интенсивному рассеиванию и перемешиванию информации в процессе вычисления значений хеш-функции. Стоит также отметить интересную особенность алгоритма, которая заключается в том, что параллельные вычисления, например, с использованием гибридных суперкомпьютерных технологий, при получении хеш-образа затруднены. Это свойство в некоторых ситуациях, например, при организации парольных систем разграничения доступа, не является лишним.

Часть 2 будет посвящена вопросам практического использования хеш-функций.

Список литературы

1. G. Brassard, editor. Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology. Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings, volume 435 of Lecture Notes in Computer Science. Springer, 1990.
2. B. Denton, R. Adhami. Modern Hash Function Construction. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.218.1717.
3. B. Preneel The First 30 Years of Cryptographic Hash Functions and the NIST SHA-3 Competition.
4. A. Rimoldi. An introduction to Hash functions. http://www.science.unitn.it/~sala/BunnyTN/rimoldi.pdf.
5. S. Al-Kuwari, J. H. Davenport, R. J. Bradford. Cryptographic Hash Functions: Recent Design Trends and Security Notions. Short Paper Proceedings of 6th China International Conference on Information Security and Cryptology (Inscrypt '10). 2010, Science Press of China, pp. 133-150.
6. J.-S. Coron, Y. Dodis, C. Malinaud, P. Puniya. Merkle-Damgård Revisited: How to Construct a Hash Function. In Advances in Cryptology — CRYPTO 2005,

volume 3621 of Lecture Notes in Computer Science, pages 430–448. Springer, 2005.

7. Announcing the ADVANCED ENCRYPTION STANDARD (AES). Federal Information Processing Standards Publication 197, November 26, 2001. http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf
8. P. Barreto, V. Rijmen. The WHIRLPOOL Hashing Function. http://cryptospecs.googlecode.com/svn/trunk/hash/specs/whirlpool.pdf
9. Информационная технология. Криптографическая защита информации. Функция хеширования. ГОСТ Р 34.11-2012. – Москва, Стандартинформ, 2012.
10. O. Kazymyrov. Prototype of Russian Hash Function "Stribog". https://www.frisc.no/wp-content/uploads/2012/05/Finse2012_Kazymyrov-Prototype_of_russian_hash_function_stribog.pdf
11. O. Kazymyrov, V. Kazymyrova. Algebraic Aspects of the Russian Hash Standard GOST R 34.11-2012. http://eprint.iacr.org/2013/556.pdf
12. P. A. Lebedev. Comparison of Old and New Cryptographic Hash Function National Standards of Russian Federation on CPUs and NVIDIA GPUs. http://paco2012.ipu.ru/procdngs/F108.pdf
13. G. Bertoni, J. Daemen, M. Peeters, G. Van Assche. Keccak specifications http://keccak.noekeon.org/Keccak-specifications-2.pdf
14. E. Biham, O. Dunkelman. A framework for iterative hash functions - HAIFA. Cryptology ePrint Archive, Report 2007/278, 2007. http://eprint.iacr.org/.
15. B. Burr. NIST Hash Competition: Where we are and what we're learning. http://middleware.internet2.edu/idtrust/2010/slides/09-burr-hashcompetition.pdf
16. T. Peyrin. State-of-the-art of Hash Functions. http://www1.spms.ntu.edu.sg/~ccrg/WAC2010/slides/session_4/4_3_Peyrin_hash.pdf .
17. S. Lucks. Design Principles for Iterated Hash Functions. http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf
18. G. Bertoni, J. Daemen, M. Peeters, G. Van Assche. Permutation-based encryption, authentication and authenticated encryption. http://keccak.noekeon.org/KeccakDIAC2012.pdf
19. G. Bertoni, J. Daemen, M. Peeters, G. Van Assche. Sponge functions. Ecrypt Hash Workshop, Barcelona, Spain, May 2007. http://sponge.noekeon.org/.
20. G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. On the indifferentiability of the sponge construction. In Advances in Cryptology —

EUROCRYPT 2008, volume 4965 of Lecture Notes in Computer Science, pages 181–197. Springer, 2008.

21. G. Bertoni, J. Daemen, M. Peeters, Van Assche, R. Van Keer. Keccak implementation overview. http://keccak.noekeon.org/Keccak-implementation-3.2.pdf
22. E. B. Kavun, T. Yalcin. A Lightweight Implementation of Keccak Hash Function for Radio-Frequency Identification Applications. http://emsec.ruhr-uni-bochum.de/media/crypto/veroeffentlichungen/2011/10/14/keccakpaper.pdf.
23. A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications. NIST Special Publication 800-22. http://csrc.nist.gov/groups/ST/toolkit/rng/documents/SP800-22b.pdf.
24. И.В. Чугунков. Методы и средства оценки качества генераторов псевдослучайных последовательностей, ориентированных на решение задач защиты информации. – М.: НИЯУ МИФИ, 2012.