УДК 004.021

СОВРЕМЕННЫЕ СТАНДАРТЫ АЛГОРИТМОВ ХЭШИРОВАНИЯ

К.С. ГОЛОВАН
(Представлено: Е.Р. СУХАРЕВ)

Рассматриваются принципы построения хэш-функций. Представлена конструкция Меркла – Дамгарда и её модификации Wide Pipe, Fast Wide Pipe, HAIFA. Описывается последняя разработка в области построения хэш-функций – криптографическая губка. Приведены примеры современных стандартов алгоритмов хэширования и их особенности.

Функция хэширования – это функция, которая принимает на вход строку битов (байтов) произвольной длины и выдаёт результат фиксированной длины. Криптографические функции хэширования играю важную роль в современной криптографии, особенно в области аутентификации сообщений, контроля целостности, цифровой подписи и хранения ключей.

Рассмотрим основные принципы построения хэш-функций.

Структура Меркла – Дамгарда описана в докторской диссертации Ральфа Меркла. Ральф Меркл и Иван Дамгор независимо показали, что если функция сжатия устойчива к коллизиям, то и хеш-функция будет также устойчива. Чтобы доказать устойчивость структуры, Меркл и Дамгор предложили дополнить сообщение блоком, который кодирует длину первоначального сообщения. Это называется упрочнение Меркла – Дамгарда.

Суть конструкции заключается в итеративном процессе последовательных преобразований, когда на вход каждой итерации поступает блок исходного текста и выход предыдущей итерации. Входное сообщение x разбивается на t одинаковые по длине блоки, длина блока x_i равна r. Длина блока r должна соответствовать длине входного блока функции сжатия f. Последний блок дополняется нулями, если его длина не равна r. К этим блокам добавляется дополнительный (t + 1)-й блок, содержащий информацию о длине сообщения x. На каждой итерации к блоку x_i применяется функция сжатия f, выходным значением которой является промежуточное значение H_i. Это значение используется в следующей итерации, поэтому предполагается использование начального вектора H_0. После обработки всех блоков к последнему промежуточному значению применяется функция финализации g. Она используется для уменьшения размера хэша или для лучшего смешивания битов. Структура Меркла – Дамгарда представлена на рисунке 1.

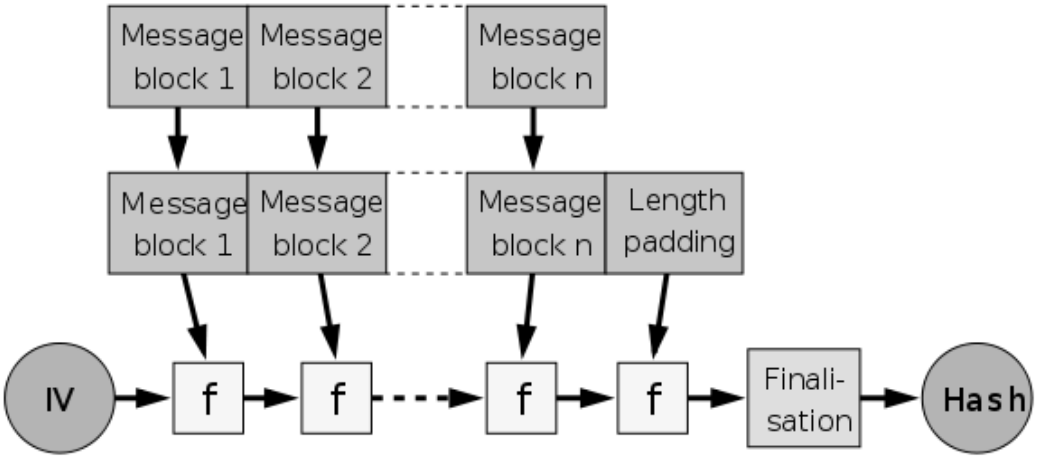

Рисунок 1. – Конструкция Меркла – Дамгарда

В качестве функции сжатия может использоваться блочный шифр. Данное расширение конструкции легло в основу схем Девиса – Мейера, Матиса – Мейера – Осеаса и Миагучи – Пренеля.

Конструкция Меркла – Дамгарда [1] используется в таких широко известных функциях, как MD5, SHA-1, SHA-2. Популярность структуры Меркла – Дамгарда обусловлена тем, что, как было доказано, если односторонняя функция сжатия f устойчива к коллизиям, то и хеш-функция, построенная на её основе, будет также устойчива к коллизиям. Несмотря на популярность схемы Меркла – Дамгарда, ряд работ показал недостатки данной конструкции, связанные с множественными коллизиями [2], дополнением сообщения до нужной длины [3], нахождением второго прообраза [4]. Атаки направлены не на конкретный алгоритм, а применимы к любой хеш-функции, построенной по схеме Меркла – Дамгарда. Атака,

основанная на множественных коллизиях [2], может быть осуществлена злоумышленником, даже если он имеет доступ к функции хеширования в целом, но не имеет возможности контролировать значения функции сжатия. Дополнение сообщения до нужной длины направлено на схему Девиса – Мейера, однако никак не затрагивает особенности самого блочного шифра, что делает эту атаку также достаточно общей. В связи с этим стали появляться предложения по модификации столь популярной конструкции.

Стефан Лакс предложил способ использования wide pipe конструкцию вместо конструкции Меркла – Дамграда. Wide pipe похож на конструкцию Меркла – Дамграда, но в данном методе длина промежуточного состояния в битах больше, чем длина хэша. Поэтому на последнем этапе функция финализации сжимает последнее промежуточное значение до нужного размера. После этого была предложена модификация этой конструкции Fast Wide Pipe. Основная идея этого метода в том, что промежуточное значение делится пополам. Одна половина идет на вход функции сжатия и складывается по модулю 2 со второй половиной. Данный метод работает быстрее, но при этом приходится использовать дополнительную память.

В 2006 году была предложена еще одна известная модификация конструкции Меркла – Дамграда под названием «HAIFA» [5]. Принципиальное отличие заключается в том, что на каждой итерации к блоку добавляется «соль» и количество бит, уже поступавших на вход функции сжатия на предыдущих итерациях. Добавление количества бит, уже поступавших на вход функции сжатия, позволяет противодействовать атакам с использованием неподвижных точек. Использование «соли» обеспечивает стойкость к атакам с предварительными вычислениями (словарная атака, использование радужных таблиц).

В 2007 году была предложена совершенно новая конструкция построения хэш-функций. Криптографическая губка [6] была разработана с целью заменить устаревшую конструкцию Меркла – Дамграда. Функция губки относится к классу алгоритмов с конечным внутренним состоянием, на вход которой поступает двоичная строка произвольной длины и которая возвращает двоичную строку также произвольной длины.

Губка – это итеративная конструкция для создания функции с произвольной длиной на входе и произвольной длиной на выходе на основе преобразований f. Губка имеет внутреннее состояние S с данными фиксированного размера b (бит). При этом данные разделены на 2 части – первая $S1$ размера r, а вторая $S2$ – размера c. Значение r называется битовой скоростью, а значение c – мощностью $b = r + c$.

На фазе инициализации блок данных размера b заполняется нулями, а входные данные M разбиваются на блоки размера r. Дальнейшая работа губки производится в 2 этапа:

1) фаза «впитывания» (absorbing) – выполняется операция XOR очередного блока исходного сообщения с первой частью состояния $S1$ размера r (бит), оставшаяся часть $S2$ состояния ёмкостью c остается незатронутой. Результат помещается в $S1$, а затем состояние S обрабатывается функцией f – многораундовой бесключевой псевдослучайной перестановкой, и так повторяется до исчерпания блоков исходного сообщения;

2) фаза «выжимания» (squeezing) – состояние S подаётся на функцию f, после чего часть $S1$ подаётся на выход. Эти действия повторяются до тех пор, пока не будет получена последовательность нужной длины (например, длины хэша).

Последние биты c зависят от входных блоков лишь опосредованно и не выводятся в ходе фазы «выжимания» (squeezing). Конструкция представлена на рисунке 2.

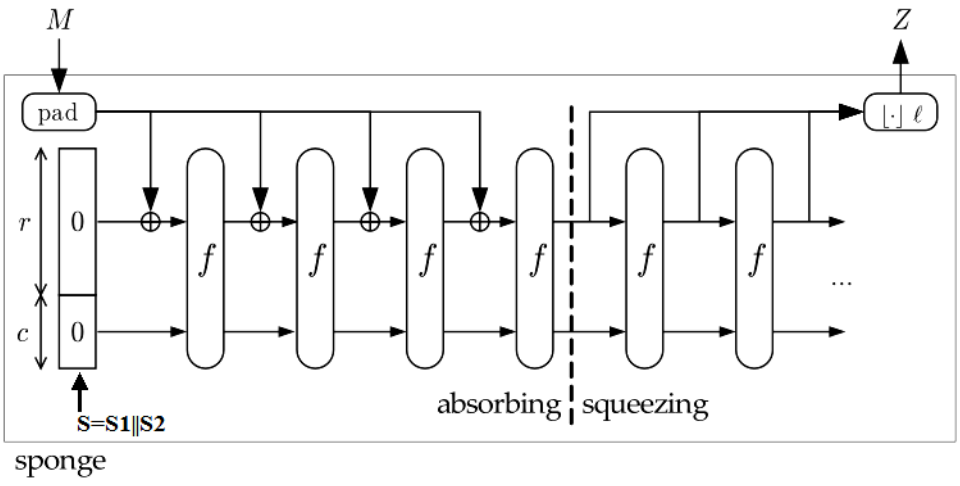

Рисунок 2. – Схема работы криптографической губки (Sponge)

Традиционный дизайн хэш-функций основан на функции сжатия. Такая функция отображает строку бит длиной m в строку длиной $m > n$ псевдослучайным образом, при этом значение n не больше 512 бит. Эта функция повторяется на протяжении нескольких раундов с различными константами, таким образом достигая нужной стойкости. Хэш-функции, построенные таким образом, преследует проблема коллизий 1 и 2-го рода. Казалось бы, многораундовое использование функции сжатия должно скрыть дефекты, но для таких функций легко находятся частичные коллизии, что ставит под сомнение стойкость всей конструкции.

Далее, рассмотрим существующие алгоритмы хэширования, которые построены с использованием конструкции Меркла – Дамграда и её модификаций.

Международный стандарт ISO/IEC 10118-3 описывает специальные функции хэширования. Функции хэширования в данном стандарте основаны на повторном использовании тактовой функции. Указывается семь отдельных тактовых функций, от которых происходят отдельные специальные функции хэширования.

Специальные функции хэширования RIPEMD-160 и SHA-1 имеют длину хэш-кода до 160 бит и начальный вектор длиной 160 бит, состоящий из пяти 32-битных определённых стандартом слов. Специальная функция хэширования RIPEMD-128 имеет длину хэш-кода до 128 бит и начальный вектор длиной 128 бит, состоящий из четырёх 32-битных определённых стандартом слов. Специальная функция хэширования SHA-256 имеет длину хэш-кода до 256 бит и начальный вектор длиной 256 бит, состоящий из восьми 32-битных определённых стандартом слов. Специальная функция хэширования SHA-384 имеет длину хэш-кода до 384 бита и начальный вектор длиной 512 бит, состоящий из восьми 64-битных определённых стандартом слов. Специальные функции хэширования SHA-512 и WHIRLPOOL имеют длину хэш-кода до 512 бит и начальный вектор длиной 512 бит, состоящий из восьми 64-битных определённых стандартом слов.

Исходя из представленных выше данных, можно сделать вывод о практическом применении данных функций хэширования: функции RIPEMD-128, RIPEMD-160 и SHA-1 не обеспечивают достаточный уровень безопасности. Используя парадокс задачи о днях рождения, можно найти реальные коллизии для данных функций за 2^{64}, 2^{80} и 2^{80} шагов соответственно.

Оставшиеся функции хэширования (SHA-256, SHA-384, SHA-512, WHIRLPOOL) являются более стойкими к атакам на основе коллизий, и на данный момент являются пригодными для практического применения. Однако в ходе проведённых исследований были проведены атаки на усечённые варианты приведённых функций. Для WHIRLPOOL сгенерировать коллизию удалось для варианта в 4.5 раунда из 10, при этом сложность данной атаки составила 2^{120}. Для функции SHA-256 были найдены множественные коллизии с вариантом в 21 раунд из 64.

В Беларуси до 2016 года были определены два стандарта: по функции хэширования СТБ 1176.1-99 [9] и СТБ 34.101.31-2011 [10].

В основе алгоритма хэширования по стандарту СТБ 1176.1-99 лежит итеративный механизм на базе четырех односторонних преобразований. Начальный вектор размером 256 бит выбирается произвольным образом, и хэш-код имеет длину 256 бит.

В основе алгоритма хэширования по стандарту СТБ 34.101.31-2011 лежит итеративный механизм на базе двух блочных односторонних преобразований, использующих шифрующую функцию, определённую в этом же стандарте. Начальный вектор размером 256 бит определён заранее, хэш-код имеет длину 256 бит.

После того как для семейства SHA-2 были приведены теоретические атаки, в 2007 году NIST было принято провести конкурс SHA-3 для выбора нового стандарта. С этого момента начался активный отбор кандидатов на использование хэш-функции в качестве нового стандарта SHA-3. Новый стандарт SHA-3 должен поддерживать семейство алгоритмов, реализующих хеш-суммы длиной 224, 256, 384 и 512 бит. Как минимум, одна функция из семейства должна поддерживать хеш-код аутентификации сообщений (HMAC) и рандомизированное хеширование. Кроме того, для всех n бит хеш-суммы алгоритм должен обеспечивать выполнение следующих условий:

- устойчивость к нахождению прообраза для n бит;
- устойчивость к нахождению второго прообраза для $(n - L)$ бит, где первый прообраз имеет длину не более $2L$ блоков;
- устойчивость к коллизиям для $n/2$ бит;
- устойчивость к атакам дополнением сообщения;
- для любого $m \leq n$ любое подмножество из m бит хеш-суммы длиной в n бит должно удовлетворять выше названным условиям.

Финалистами конкурса стали алгоритмы Skein, JH, Grostl, BLAKE и Keccak. Из них самые лучшие показатели продемонстрировал алгоритм Keccak, в 2012 году став новым стандартом хэширования SHA-3. Алгоритм 5 августа 2015 года утверждён и опубликован в качестве стандарта FIPS 202. Алгоритм SHA-3

построен по принципу криптографической губки (данная структура криптографических алгоритмов была предложена авторами алгоритма Keccak ранее). Функция сжатия представляет собой 1600-битовую перестановку. Размер блока сообщения зависит от желаемого размера хеш-суммы: для Keccak-512, -384, -256, -224 блок сообщения имеет размер 576, 832, 1088 и 1152 бит соответственно. Также команда разработчиков Keccak предложила версии с уменьшенной длиной перестановки в 25, 50, 100, 200, 400 и 800 бит вместо 1600 бит. Однако в качестве стандарта была принята только функция Keccak с длиной состояния 1600 бит, и разработчики рекомендуют пользоваться только ей. Функция сжатия f перемешивает внутренне состояние алгоритма, которое имеет вид массива 5×5 с элементами длиной 64 бита, на протяжении 24 раундов.

Атаки на нахождение коллизий (двух произвольных сообщений, дающих одинаковый хэш) и вторых прообразов (сообщения, дающего такой же хэш, что и имеющееся) имеют важное практическое и теоретическое значение, но наряду с неинвертируемостью не обеспечивают полной оценки стойкости хэш-функции. Существует целый ряд экзотических атак – на удлинение сообщения, атаки на частичные коллизии с подобранным префиксом и др. Чтобы доказать стойкость конструкции губки ко всем возможным атакам, авторы приняли ещё одно радикальное решение – считать единственным и достаточным общим критерием стойкости неразличимость хэш-функции от случайного оракула [7].

Случайный оракул (*Random Oracle*) – это идеализированная функция, описывающая работу машины с практически бесконечным объёмом памяти, которая на любой запрос может выдать идеально случайное число и запомнить пару «запрос – ответ». Если такой же запрос будет когда-либо повторён, то ответ будет не генерироваться заново, а взят из запомненного списка. Если перестановка f, лежащая в основе губки идеальна, то и хэш-функция доказуемо неразличима от RO, значит никакие из возможных атак действовать не будут. Конечно, есть оговорка на наличие универсального тривиального различителя для всех возможных хэш-функций: построение случайного оракула на алгоритме с конечным числом состояний невозможно. Например, коллизия внутреннего состояния приведёт к неслучайному выходу, что было бы невозможно в RO, у которого никаких внутренних состояний нет [7].

Авторы Keccak доказали, что стойкость такой конструкции не отличается от случайного оракула с размером выхода равным $c/2$, при условии, что f – идеальная функция перестановки. То есть чтобы получить хэш с доказанной математически стойкостью в 512 бит, нужно сделать размер параметра c (независимой части состояния) 512*2 = 1024 бита.

Данные теоретические результаты позволяют использовать Keccak в качестве хэш-функции с произвольным размером выхода, потокового шифра, функции выработки ключей из пароля, кодов аутентификации сообщений, криптостойкого генератора псевдослучайных чисел с подкачкой энтропии из внешнего источника и с затиранием внутреннего состояния.

В 2014 году появились первые атаки на Keccak [8]. В режиме хэш-функции взлом пока составляет 5 раундов, в ключевых режимах – 9 раундов. Но для полных 24 раундов исследователи оценивают стойкость ключевых режимов Keccak как всё ещё высокую.

В 2016 году в Беларуси появился новый стандарт алгоритмов хэширования СТБ 34.101.77 [12]. Переход от алгоритма СТБ 34.101.31 к алгоритмам этого стандарта позволяет увеличить скорость хэширования, по крайней мере, на паритетном уровне стойкости и на 64-разрядных аппаратных платформах. Кроме этого, алгоритмы хэширования данного стандарта поддерживают все три уровня стойкости алгоритмов ЭЦП, определённых в стандарте СТБ 34.101.45 [11], в то время как алгоритм СТБ 34.101.31 поддерживает только первый уровень. В отличие от Keccak, СТБ 34.101.77 внутренне состояние имеет длины 1536. Состояние представляется в виде массива 3×8 с элементами длиной 64 бита. Преобразование выполняется, так же как и в Keccak, на протяжении 24 раундов, при этом в каждом раунде происходит наложение раундовой константы с помощью операции XOR. В целом пошаговая функция f СТБ 34.101.77 разработана иначе. На каждом раунде в преобразовании дополнительно участвуют 5 переменных.

Алгоритмы этого стандарта отличаются уровнем стойкости l. Это натуральное число, кратное 16 и не превосходящее 256. Алгоритм уровня l вычисляет хэш-значения длины, обрабатывая входные слова блоками длины $1536 – 4l$. Уровни $l = 128$, $l = 192$, $l = 256$ являются стандартными. Разработчики стандарта рекомендуют отдавать им предпочтение.

Заключение. Криптография не стоит на месте и развивается. Функции хэширования используются во многих программных средствах, что повышает требования к их стойкости. Принятие нового стандарта алгоритмов хэширования SHA-3 стимулировало развитие различных областей криптографии, а также послужило толчком к разработке модификаций и новых алгоритмов.

ЛИТЕРАТУРА

1. Merkle, R.C. A Certified Digital Signature / R.C. Merkle // In Advances in Cryptology – CRYPTO’89 Proceedings // Lecture Notes in Computer Science. – Vol. 435; ed. G. Brassard; Springer-Verlag, 1989. – P. 218–238.

2. Joux, A. Multicollisions in iterated hash functions. Application to cascaded construction / Antoine Joux // In Advances in Cryptology – CRYPTO '04 Proceedings, Lecture Notes in Computer Science, Vol. 3152, M. Franklin, ed, Springer-Verlag, 2004. – 306 – 316 p.
3. Kelsey, J. A long-message attack on SHAx, MDx, Tiger, N-Hash, Whirlpool, and Snefru. Draft / John Kelsey. – Unpublished Manuscript.
4. Maurer, U. Indiff erentiability, Impossibility Results on Reductions, and Applications to the Random Oracle Methodology / Ueli Maurer, Renato Renner, Clemens Holenstein // In M. Naor, editor, Theory of Cryptography, First Theory of Cryptography Conference, Vol. 2951 of Lecture Notes in Computer Science. – Springer, 2004. – P. 21–39.
5. Biham E. A Framework for Iterative Hash Functions – HAIFA / Eli Biham, Orr Dunkelman // Cryptology ePrint Archive, 2007 [Electronic resource]. – Mode of access: http://eprint.iacr.org/2007/278. – Дата доступа: 25.09.2016.
6. Cryptographic sponge functions – STMicroelectronics / Guido Bertoni [et al.], 2011. – 93 с.
7. Проект OpenPGP в России [Электронный ресурс] / Хэш-функция Keccak и конструкция Sponge как универсальный криптопримитив. – Россия, 2007. – Режим доступа: https://www.pgpru.com/biblioteka/statji/keccaksponge. – Дата доступа: 25.09.2016.
8. Проект OpenPGP в России [Электронный ресурс] / Низкая алгебраическая сложность Keccak/Keyak облегчает кубические атаки. – Россия, 2007. – Режим доступа: https://www.pgpru.com/novosti/2014/nizkajaalgebraicheskaja slozhnostjkeccakkeyakoblegchaetkubicheskie ataki. – Дата доступа: 25.09.2016.
9. Информационная технология. Защита информации. Функция хэширования: СТБ 1176.1-99. – Минск : Госстандарт, 1999.
10. Информационные технологии. Защита информации. Криптографические алгоритмы шифрования и контроля целостности: СТБ 34.101.31-2011. – Минск : Госстандарт, 2011.
11. Информационные технологии. Алгоритмы электронной цифровой подписи и транспорта ключа на основе эллиптических кривых : СТБ 34.101.45 – 2013. – Минск : Госстандарт, 2013.
12. Информационные технологии. Защита информации. Алгоритмы хэширования: СТБ 34.101.77-2016. – Минск : Госстандарт, 2016.