УДК 681.3.06

Е.В. КОТУХ, В.М. КАРТАШОВ, д-р техн. наук, О.Г. ХАЛИМОВ,
Д.П. ЦАПКО, А.В. САМОЙЛОВА

АНАЛИЗ СОВРЕМЕННЫХ ТРЕБОВАНИЙ К КРИПТОГРАФИЧЕСКИМ ПРИМИТИВАМ НОВОГО ПОКОЛЕНИЯ

Введение

В первом квартале 2015 г. компания Google официально прекратила поддержку SHA-1 [1]. Практически, это означает, что с выходом браузера Chrome версии 41 все сайты, подписанные сертификатом с указанным алгоритмом, будут признаваться небезопасными. В свою очередь, компания Microsoft анонсировала отказ от 160-битных хешей с 1 января 2017 г. [2]. Это стало возможным благодаря значительному повышению производительности в совокупности с новыми методологиями распределительных вычислений, что позволило реализовать известные и новые угрозы безопасности, характеризуемые снижением фактических временных затрат на преодоление практической стойкости подсистемы аутентификации до неприемлемого уровня. Два последних десятилетия активно разрабатывались математические методы, представляющие практический интерес для криптоанализа. Вычислительная сложность решения определенного класса математических задач лежит в основе безопасности ряда систем защиты коммерческих информационно-коммуникационных технологий.

В данной связи одной из актуальных научных задач является разработка перспективных криптопримитивов с доказуемой стойкостью. Цель статьи – исследование безопасности и производительности современных криптопримитивов-финалистов конкурса NIST, анализ причин выбора криптопримитива Keccak в качестве финалиста, современных требований к безопасности, вычислительной сложности и скорости, методов повышения скорости без увеличения сложности реализации.

Влияние вычислительной мощности на требования к хеш-функциям

В качестве основных составляющих прироста мощности вычислительных систем можно выделить: рост производительности выделенных вычислительных устройств; организацию массовых вычислений в совокупности устройств, в том числе и мобильных; использование эффективных моделей вычислений на существующих классах архитектур. Одним из наиболее перспективных направлений в решении задач вычислений общего назначения является использование технологии GPGPU (General-purpose graphics processing units) [3, 4]. Графический процессор (GPU) обладает меньшим набором исполняемых команд (RISC-подобные архитектуры), чем CPU, но большей производительностью. Технология GPGPU позволяет на одном вычислителе достигать достаточно высокого уровня параллелизма без временных затрат на передачу данных между узлами и синхронизацию результатов вычислений. Относительно низкая стоимость, простота добавления вычислительных модулей и удельное энергопотребление в сочетании с высокой удельной производительностью GPU позволяют реализовать на практике массовые распределенные параллельные вычисления, которые доступны более широкому кругу потенциальных нарушителей. Сравнительный анализ возможностей CPU и GPU архитектур в практической реализации алгоритмов SHA1 и MD5 представлен на рис. 1.

Слабая вычислительная сложность, ряд значительных недостатков конструкции и математически обоснованные атаки на коллизионную стойкость позволили реализовать для хеш-функций SHA1 и MD5 атаки полного перебора за допустимое время. В практической реализации [14] представлены результаты скорости подбора MD5-паролей на nVidia TitanX (около 135,2 миллиардов комбинаций в секунду), что позволяет найти пароль длиной восемь символов менее чем за пять минут.

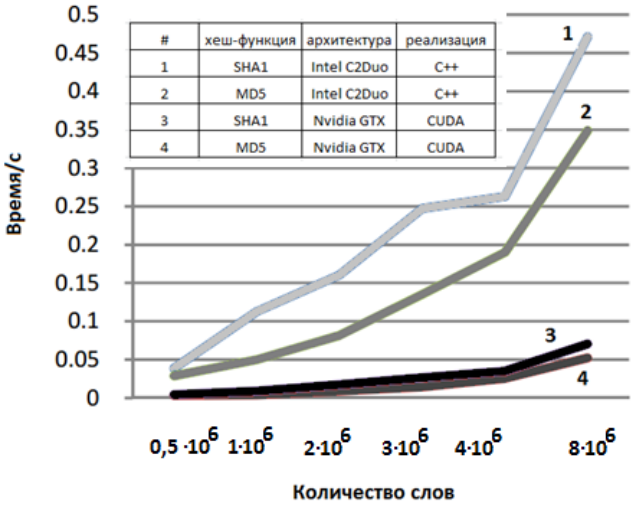


Рис. 1. Сравнительный анализ скорости реализации хеш-функций на CPU/GPU архитектурах

В табл. 1 представлены результаты по производительности реализации атаки «полного перебора» для семейства хеш-функций на тестовом стенде (Ubuntu 14.04, 64 bit, ForceWare 346.29, 8x NVidia Titan X, stock core clock, oclHashcat v1.3)

Таблица 1

Производительность атак полного перебора для различных хеш-функций

Алгоритм	Производительность (комбинаций/с)
MD5	$135232 * 10^6$
SHA1	$42408 * 10^6$
SHA256	$16904 * 10^6$
SHA512	$5240 * 10^6$
RipeMD-160	$28368 * 10^6$
Whirpool	$1122402 * 10^6$

Успешные атаки и рост вычислительной емкости предъявляют высокие требования к стойкости хеш-функций. Недостатки, выявленные у криптографических стандартов 90-х, и обоснованные теоретические атаки обусловливают развитие новых методов реализации криптопримитивов. Конкурс SHA-3, организованный NIST, показал развитие требований к безопасности, архитектуре и производительности при реализации криптографических примитивов. В финале конкурса SHA-3, организованного NIST, двое из пяти финалистов (Keccak и Skein) оказались универсальными криптопримитивами, которые могут использоваться не только для хеширования, но и для выполнения множества других криптографических операций, обеспечивая упрощение проектируемых криптографических протоколов. Три этапа конкурса фактически формализировали дальнейший подход к процедуре выбора криптопримитивов. Рассмотрим требования к безопасности, архитектуре и производительности, предъявляемые к современным криптопримитивам.

Требования к безопасности современных хеш-функций

Основные определения безопасности хеш-функций представлены в [7].

Определение 1 (Стойкость к вычислению прообраза) *Хеш-функция* h: $\{0,1\}^* \rightarrow R$ *является стойкой к вычислению прообраза силой* (t, ϵ), *если не существует вероятностного алгоритма* I_h, *с входными значениями* $Y \in_R R$ *и значениями на выходе* $X \in \{0,1\}^*$, *временем выполнения не более чем* t, *где* $h(X) = Y$ *и вероятностью не менее* ϵ, *оцененной при случай-*

ном выборе Y и I_h. Стойкость хеш-функций к вычислению прообраза имеет важное значение для систем аутентификации, использующих хэш-значения паролей и секретных ключей.

Определение 2 (Стойкость к вычислению второго прообраза). *Пусть S – конечное подмножество* {0,1}*. *Хеш-функция h:* {0,1}*→ *R является стойкой к вычислению второго прообраза силой (t, є, S), если не существует вероятностного алгоритма S_h, с X $Є_R$ S и X′ Є* {0,1}*, *временем выполнения не более чем t, где X′ ≠ X и h (X′) = h (X) и вероятностью не менее є, оцененной при случайном выборе X и S_h.* Стойкость хеш-функций к вычислению второго прообраза определяет безопасность систем аутентификации с цифровой подписью.

Определение 3 (Стойкость к коллизиям). *Хеш-функция h*: {0,1}* → *R является стойкой к коллизиям силой (t,* є*), если не существует вероятностного алгоритма C_h с известными выходными значениями X, X′ Є* {0,1}*, *временем выполнения не более чем t, где X′ ≠ X и h (X) = h (X′) и вероятностью не менее є, оцененной при случайном выборе Ch.*

Основные требования к безопасности хеш-функций, используемые NIST, представлены в работе [6]. Сравнительный анализ финалистов конкурса NIST, удовлетворяющих основным требованиям к безопасности, приведен в табл. 2.

Таблица 2

Безопасность финалистов конкурса NIST

Кандидат	Стойкость к коллизиям	Кол-во раундов сжатия	Сложность вычисления		
			прообраза	второго прообраза	PRF
Blake-256	Inner-collision	2.5	2^{224}	2^{256}	-
Blake-512	Inner-collision	4	2^{448}	2^{256}	-
Grostl-256	2^{64}	5	2^{256}	2^{256}-2^{512}	$2^{244.85}$
Grostl-512	2^{128}	8	2^{512}	2^{512}-2^{1024}	2^{248}
JH-256	$2^{96,12}$	16	-	$2^{\sim 388}$	-
JH-512	$2^{95,63}$	22	-	$2^{\sim 900}$	-
Keccak-224	~2^{256}	4-5	2^{112}	2^{288}	
Keccak-256	~2^{256}	5-10	2^{1370}	2^{512}	
Keccak-512	2^{512}	24	2^{1590}	$2^{511.5}$	2^{1576}
Skein-256	>$2^{\sim 265}$	32-36	2^{105}	2^{200}-2^{824}	$2^{511.7}$
Skein-512	>$2^{\sim 265}$	32-36	2^{105}	2^{200}-2^{824}	$2^{511.7}$
Skein-1024	>$2^{\sim 265}$	32-36	2^{105}	2^{200}-2^{824}	2^{1045}-2^{125}

Более 40 кандидатов не удовлетворили этим определениям и были исключены в первом раунде конкурса [8].

Требования к архитектуре и реализации

Обобщим требования к архитектуре и особенности реализации, являющиеся частью интегральной оценки алгоритма. Функция сжатия как основной элемент архитектуры хеш-функции была заявлена в большинстве кандидатов. С точки зрения требований к архитектуре, важными аттрибутами являются: особенности реализации структуры SPN и блоков подстановок (S-box) или блоков перестановок (P-box), схемы Фейстеля для преобразований функций $F(L_i,K_i)$, математическая сложность функции ключевого расширения (функции разворачивания подключей раунда из основного ключа), структуры Merkle-Damgard, Wide Pipe, размерность MDS-матрицы. В работе [9] рассмотрено влияние булевых операций, OUT-трансформации, FSR, ARX-сдвигов на конечную реализацию хеш-функций.

С точки зрения оптимизации вычислений, наиболее важным параметром является *размер кэша* 1-го уровня для инструкций. Кэш 1-го уровня для данных важен для функций, которые используют таблицы собственных имплементаций (такие, как хеш-функция ECHO). Несоответствие количества выполняемых для цикла инструкций предполагает серьезное снижение производительности (примером являются некоторые реализации Skein), где впоследствии предполагается использовать метод «размотки цикла» [10]. Этот метод, который

заключается в повторении кода для одного раунда нескольких последовательных раундов, позволяет «обнулить» затраты на маршрутизацию данных в памяти. Так, например, в алгоритме SHA-256 в каждом раунде «вращаются» восемь слов, что для восьми последовательных раундов «размотки цикла» дает возможность использовать те же переменные и сократить затраты на копирование данных между ними. Кроме того, если «разматывать» 64 последовательных раунда SHA-256, то можно отказаться от получения констант из таблицы и сэкономить на косвенной адресации в памяти. Такой подход является общим инструментом уменьшения затрат на маршрутизацию данных во время выполнения. Если реализация в процессе полной «размотки» помещается в кэш 1-го уровня, то она оптимизирована, иначе возникают дополнительные затраты на копирование и косвенную адресацию. Важной особенностью реализации является *возможность использовать 64-разрядные целые числа* для систем с собственными 64-разрядными регистрами [10]. На 32-битных системах без таких регистров использование 64-битных целых крайне неэффективно и требует два регистра. Это увеличивает затраты на коды операций (перенос между нижними и верхними словами) для 32-битных архитектур. Наиболее очевидно это проявляется на диаграммах с 512-битными хеш-кодом, полученным на 64-битных архитектурах. В архитектуре x86 отсутствие 64-разрядного целого типа часто компенсируется в реализациях хэш-функций с использованием специальных блоков с 64-разрядными регистрами (MMX, SSE). Недостатком реализаций в подобных случаях является необходимость использовать особенные, встроенные в компилятор инструкции (C/C++) [10]. Это ограничивает возможности реализации на других платформах, в частности в Java VM. Порядок байт для современных алгоритмов хеш-функций уже не имеет существенного значения для достижения высокой производительности.

Для большинства функций текущий набор инструкций также не важен. В больших системах коды операций динамически транслируются во внутренние элементарные инструкции, для которых CPU применяет оптимизации (параллельное выполнение, изменение порядка, спекулятивное выполнение инструкций и т.д.).

Большинство из кандидатов NIST второго раунда соревнований показали композитную схему реализации. Некоторые из них были представлены парой функций для разной длины выходов (224,256,384,512- бит). Fugue и Lufa состояли из трех функций, а Keccak – из четырех, хотя и с разделяемым ядром. В свою очередь, такой подход в архитектуре имеет ряд негативных последствий для производительности: реализация семейства функций требует больше ресурсов для разработки, чем для оптимизации; увеличивается размер кода; проблемы производительности и безопасности более не являются взаимозависимыми. Однако архитектура CubeHash, JH и Shabal позволила избежать подобного эффекта и показала стабильную производительность для всех размеров выходных.

Минимальный размер ключа, необходимый для защиты информации от атак злоумышленника, будет расти по мере повышения быстродействия компьютеров, тем не менее, приведенные вычисления показывают, что существует возможность выбрать такую длину ключа, что атаку методом полного перебора провести будет в принципе невозможно, вне зависимости от повышения вычислительной мощности компьютеров или успехов в области классической теории алгоритмов

В работе [11] проанализирован подход к классификации современных хеш-функций на основе сравнения с эталонной реализацией алгоритма SHA-256/512 в NIST (Intel Core 2 Duo) (табл. 3). По результатам конкурса класс скорости в работе [11] был определен следующим образом: AA=x<1/2 *SHA-2*; A=1/2 *SHA*-2≤x<3/4*SHA-2*; B = 3/4 *SHA*-2≤x<*SHA*-2; C = *SHA*-2≤x<5/4*SHA*-2; D== 5/4 *SHA*-2≤x≤2*SHA*-2; E=x>2*SHA*-2.

Проанализируем особености архитектуры и реализации финалистов NIST, их влияние на вычислительную сложность, скорость и безопасность.

Таблица 3

Сравнительный анализ скорости хеш-функций

Хеш-функция	Производительность 32-бит		Производительность 64-бит	
	cpb	класс скорости	cpb	класс скорости
SHA-256	29.3	C	20.1	C
SHA-512	55.2	C	13.1	C
Blake-256	28.3	B	16.7	B
Blake-512	61.7	C	12.3	B
Grostl-256	22.9	B	22.4	D
Grostl-512	37.5	A	30.1	E
JH-256	21.3	B	16.8	B
JH-512	21.3	AA	16.8	D
Keccak-256	35.4	C	10.1	A
Keccak-512	68.9	C	20.3	D
Skein-256	21.6	A	7.6	AA
Skein-512	20.1	AA	6.1	AA

Анализ схем реализации финалистов. Основные результаты

Алгоритм BLAKE. Представлен в качестве альтернативы существующим SHA-2/MD5 применениям, значительно превосходя их по надежности, практически не уступая в производительности (см. рис. 2)

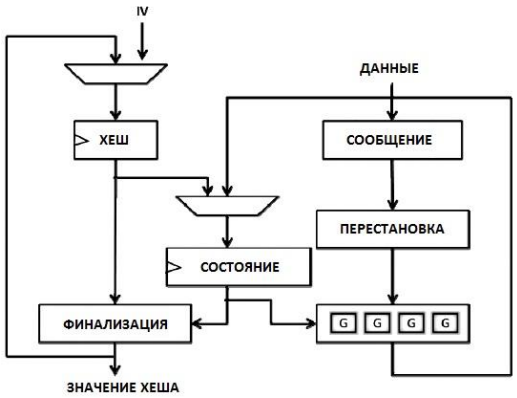


Рис. 2. Высокоуровневая архитектура BLAKE

Хеш-функция BLAKE2 не чувствительна к размеру хешируемых данных и защищена от всех свойственных SHA-1 и MD5 видов атак, связанных с возникновением коллизий в процессе хеширования. Реализации для различных архитектур с поддержкой распараллеливания позволяют существенно увеличить производительность.

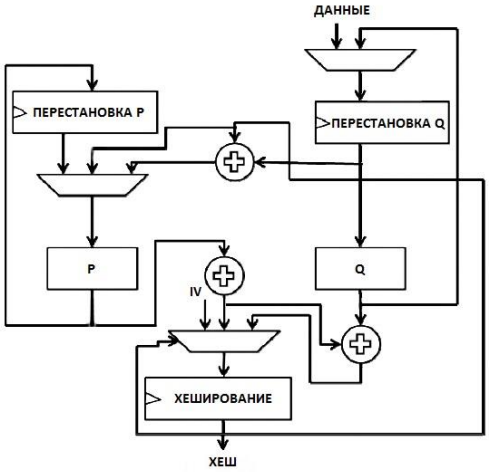


Рис. 3. Высокоуровневая архитектура Grostl

Алгоритм Grostl. Функция хэширования Grostl способна возвращать хэш-значение произвольной длины от 1 до 64 байт, то есть от 8 до 512 бит, при этом хеш-значение должно быть кратно байту (см. рис. 3). Функция сжатия базируется на двух ι-битовых перестановках P и Q [10]:

$$F(h,m)=P(h \oplus m) \oplus Q(m) \oplus h . \quad (1)$$

Выходное хеширование Ω можно описать с помощью формулы

$$\Omega(h)=trunc_n(P(x) \oplus x). \quad (2)$$

Алгоритм JH. Реализует функцию сжатия. Сообщение разбивается на блоки по 512 (хеш-значение может иметь размер 224, 256, 384 и 512 бит). Функция сжатия формирует 1024-битное значение, которое урезается до требуемого размера хэш-значения. На вход функции сжатия поступают 512-битные блоки сообщения. а на этапе финализации – 1024-битное выходное значение после обработки предыдущего блока H_{i-1} [10]. Для обработки первого блока используется значение вектора инициализации IV (рис.4).

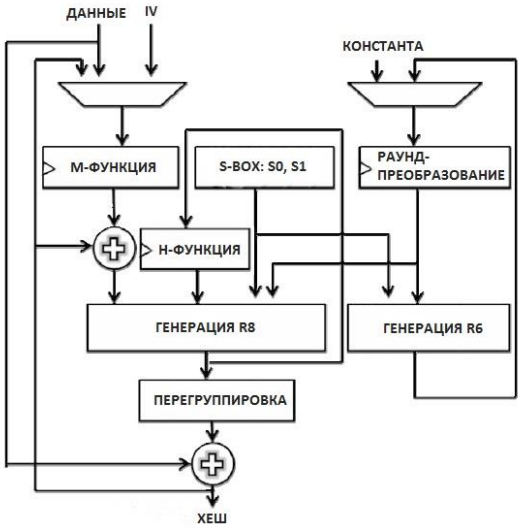


Рис. 4. Высокоуровневая архитектура JH

Функция сжатия выполняет следующие действия. Обрабатываемый блок сообщения M_i складывается по модулю два с левой 512-битной половиной значения H_{i-1}. Результат предыдущей операции обрабатывается функцией преобразования E. Сообщение M_i складывается по модулю два с правой половиной 1024-битного выходного значения функции E. В результате получается значение H_i. Выходное 1024-битное значение функции E представляется в виде восьмимерного массива, каждое измерение которого содержит два слова по четыре бита. В основе E функции лежит блочный шифр, который представляет собой SPN преобразование на основе подстановок и перестановок и состоит из 35 раундов. В каждом раунде используются: замена с помощью двух S-блоков *S0* или *S1*; линейное преобразование, поочередно обрабатывающее по два слова состояния с помощью операций XOR над определенными битами входных слов; перестановка слов состояния.

Алгоритм Skein. Функция хэширования Skein благодаря использованию нового класса блочных шифров TBC (Tweakable Block Ciphers) в совокупности с уникальной блочной итерацией (UBI) и возможностью использовать выборочную систему параметров обладает широким набором свойств. Данный подход позволил реализовать множество режимов работы, таких как: простая хэш-функция, функция древовидного хэширования, MAC-код, в качестве составной части HMAC. Skein поддерживает рандомизированное хэширование, использование в цифровых подписях, в качестве функции вычисления производных ключей (KDF), ключа из пароля (PBKDF), в качестве генератора псевдослучайных чисел (PRNG), в качестве потокового шифра [10]. Использование настраиваемого блочного алгоритма шифрования с

UBI режимом гарантирует, что каждый блок будет обработан с использованием уникальной функции сжатия. В отличие от псевдослучайных функций (Pseudo Random Function – PRF) в конструкции функций сжатия псевдослучайные перестановки (Pseudo Random Permutation – PRP), используемые в блочных шифрах, позволяют получить более быструю реализацию (достаточно использовать шифр с размером блока и размером ключа 512 бит и можно будет получить функцию сжатия m → n (m > n)). Часто конструкция блочного шифра обусловливает использование облегченного или слабого ключевого расширения. Потенциально это позволяет реализовать атаки со связанными ключами, что ослабляет надежность функции сжатия на основе блочного шифра. Идеальная функция «ключевого расширения» для идеального блочного шифра должна обладать свойствами идеальной псевдослучайной функции.

Skein защищена от новых видов специфических атак на хэш-функции – подбор удлиненных сообщений, псевдоколлизии. Вместо выбора разных схем и стандартов и изучения особенностей их применения, работы и реализации разработчикам криптоприложений можно использовать Skein и Keccak с различными параметрами. Традиционное построение хэш-функций основано на использовании функции сжатия. Эта функция отображает значение m → n (m > n) псевдослучайным образом. При этом значение n должно быть до 512 бит, а m – порядка 2n. Повторяя эту функцию в течение нескольких раундов с различными константами, достигают нужного значения стойкости. Дополняя и сцепляя между собой блоки от разных фрагментов исходного текста, получают возможность вычислить хеш от сообщения произвольной длины. Такой метод является алгоритмически сложным. Многораундовые повторения сглаживают дефектность, но наличие быстрой возможности найти частичную коллизию в исходной функции сжатия не гарантирует стойкость всей конструкции (рис. 5).

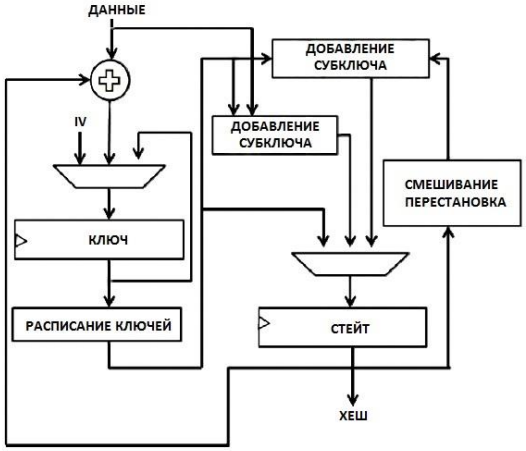


Рис. 5. Высокоуровневая архитектура Skein

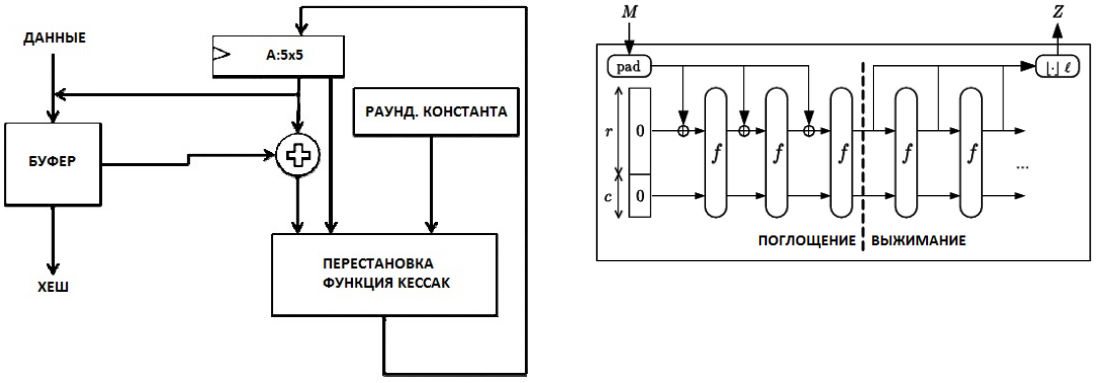
Алгоритм Keccak. Авторы алгоритма Keccak ("Кеччак") утверждают, что сконструировать надежную функцию сжатия вида m → n (m > n) как однораундовый блок криптопримитива не представляется возможным. В Keccak в качестве стойкого криптопреобразования вместо функции сжатия была реализована бесключевая PRF. Архитектурой всего алгоритма является конструкция Sponge (англ. Sponge construction), относящаяся к классу алгоритмов с конечным внутренним состоянием, на вход которой поступает двоичная строка произвольной длины и которая возвращает двоичную строку также произвольной длины *f:{0,1}^n→{0,1}^** [10]. Губка является обобщением хэш-функций, потоковых и блочных шифров, генераторов псевдослучайных чисел, имеющих произвольную длину входных данных. Простое добавление секретного ключа на вход хэш-функции Keccak превращает ее в код аутентификации сообщений. Это было невозможно в обычных хэш-функциях SHA-1 или SHA-2 и требовало громоздкой конструкции HMAC. Рассмотрим архитектуру алгоритма Keccak.

Инициализация. Массив из 5×5 строк, указывающих в направлении ось z. В официально представленной версии число двоичных элементов в строке *z* определено для вычислений

на 64-разрядных процессорах как w=64 [14]. Таким образом, состояние содержит b = 5×5×64=1600 бит. Строка S из 5×5×w бит сопоставлена с b битами состояния a следующим образом [11]:

$$s[w(5 \times y + x) + z] = a[x][y][z]. \quad (3)$$

На фазе инициализации блок данных размера b заполняется нулями, а входные данные M разбиваются на блоки размера r. Исходное сообщение M дополняется до размера блока, равного части блока $b=r+c$.



а *б*

Рис. 6. Высокоуровневая архитектура Keccak – *а*; конструкция «губка» – *б*

Смежные b биты в состоянии могут быть разделены между «внешней частью» (первые r бит) и «внутренней частью» (переменное значение c бит). Значение c выбирается как $2N$, где N – размер выходных данных, сгенерированных алгоритмом.

Фаза «поглощения». В фазе «поглощения» выполняется операция XOR очередного блока исходного сообщения с первой частью состояния размерностью r(бит), оставшаяся часть состояния размерностью c бит остается незатронутой. Результат операции и нетронутая часть передаются на вход функции Keccak-f – многораундовой бесключевой псевдослучайной перестановки и повторяется до исчерпания блоков исходного сообщения. Рассмотрим конструкцию многораундовой перестановки:

Хеш-функция Keccak-*f*. Функция f в алгоритме Keccak, представленная на рис. 6, *б*, удовлетворяет требованиям безопасности к хеш-функции. Количество раундов $R(p)$ увеличивается на размер w в соответствии с формулой p=12+2*l, где $l=log_2(w)$. При w=64 количество $R(p)$ = 24. Каждый раунд состоит из пяти перестановок состояния:

$$R = \theta \circ \rho \circ \pi \circ \chi \circ \iota. \quad (4)$$

Перестановка θ. Первая перестановка конструкции «губка» обеспечивает диффузию. Значение каждого бита в матрице состояния рассчитывается как XOR между суммами (над полем GF (2)) двух других рядов и собственного значения бита:

$$a[x][y][z] = a[x][y][z] + \sum_{y'=0}^{4} a[x-1][y'][z] + \sum_{y'=0}^{4} a[x][y'][z-1]. \quad (5)$$

Перестановка ρ. Перестановка выполняет вращение строк в состоянии с помощью предопределенной константы. Для каждого бита в состоянии в раунде n имеем:

$$a[x][y][z] = a[x][y][z + T(n)]. \quad (6)$$

Перестановка π. Перестановка при $x=y'$ и $y=2x'+3y'$ упорядочивает строки следующим образом:

$$a[x][y][z] = a[x'][y'][z]. \quad (7)$$

Перестановка χ. Перестановка является единственной нелинейной функцией. Без нее функция *Keccak-f* была бы линейным отображением над GF(2) [14]. Перестановка χ применя-

ется к каждой строке состояния, что практически реализует известную конструкцию S-boxes для каждой строки состояния:

$$a[x][y][z] = a[x][y][z] + (a[x][y][z + 1] \wedge a[x][y][z + 2]) \quad (8)$$

Перестановка ι. Перестановка предназначена для внесения ассиметрии в конструкцию за счет добавления раундовых констант, что нивелирует свойства инварианта и позволяет избежать трансляции, тем самым предупреждая слайд-атаки, использующие симметрию [14].

Фаза «выжимания». В фазе «выжимания» состояние *S* подается на функцию *f*, после чего часть *S*1 подается на выход. Эти действия повторяются, пока не будет получена последовательность нужной длины (длины хэша). Последние биты *c* зависят от входных блоков лишь опосредованно и не выводятся в ходе фазы «выжимания».

Атаки на нахождение коллизий и вторых прообразов имеют важное практическое и теоретическое значение, но наряду с неинвертируемостью не обеспечивают полной оценки стойкости хэш-функции. Существует класс атак, связанных с практическими и теоретическими уязвимостями конструкций хеш-функций: атаки на удлинение сообщения, атаки на частичные коллизии с подобранным префиксом и др. В целях доказуемой стойкости функции «губка» авторами Keccak был предложен критерий «случайного оракула» (Random Oracle) – идеализированной функции, описывающей работу идеального автомата с практически бесконечным объемом памяти, который на любой запрос выдает идеально случайное число и запоминает пару «запрос-ответ». При повторе запроса ответ не генерируется, а выдается из ранее сгенерированного массива. Если функция Keccak-f с пятью многораундовыми перестановками идеальна, то хэш-функция доказуемо неразличима от RO. Неразличимость хэш-функции от случайного оракула считается единственным и достаточным критерием стойкости. Данное утверждение позволило обосновать использование хэш-функции Keccak в качестве практически универсального криптопримитива.

Если перед блоками сообщения ввести блок с секретным ключом *K*, то получится код аутентификации сообщения (*Message Authentication Code – MAC*). Практический интерес представляет возможность вычисления MAC-кода в паралельном режиме (рис. 7).

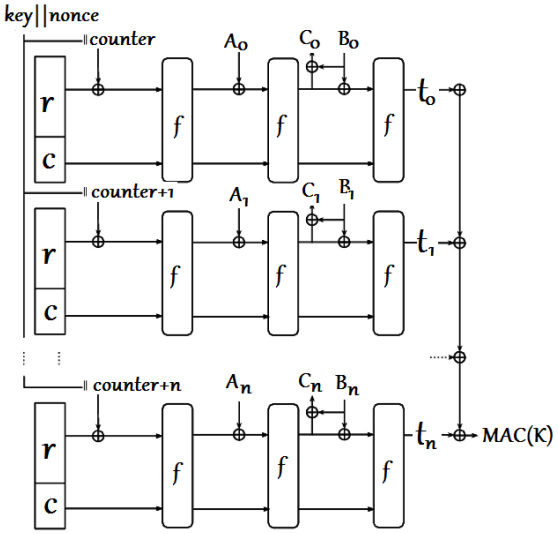


Рис. 7. Параллельное вычисление Keccak-MAC

Ранее считалось, что такие параллельные режимы возможны только для блочных шифров (OCB, GCM) или при использовании громоздкого режима древовидного хеширования. В работе [12] был предложен способ параллельного вычисления MAC-кода на основе конструкции «губка» (рис. 7). Ключ объединяется с вектором инициализации (Nonce) и подается на вход множества параллельных конструкций, где предварительно объединяется еще и со значением счетчика каждой конструкции. Возможность использовать Keccak в параллельном режиме существенно упрощает создание протоколов, требующих шифрования с аутентифи-

кацией, и избавляет от множества потенциальных ошибок в их проектировании и реализации. Это является более стойкой альтернативой патентованным и легкоуязвимым при неправильном исполнении режимам аутентифицированного шифрования на блочных шифрах (OCB, GCM). Keccak (в отличие от Skein) не содержит в своей основе блочный шифр и поэтому не является универсальным, но может использоваться в протоколах, где нужно использовать блочный шифр в режиме счетчика. Фактически Keccak может работать в четырех режимах: MAC-код, потоковый шифр, потоковый шифр с произвольным доступом, генерация симметричных ключей из паролей.

Keccak так же универсален, как и Skein, но область его применения может быть шире. При необходимости этот алгоритм может быть внедрен как в миниатюрные устройства с ограниченными ресурсами, так и в высокопроизводительные серверы, работающие с большим объемом соединений. Оба алгоритма имеют самую сильную доказательную базу среди всех финалистов, но по оценкам Keccak имеет более строгие доказательства в модели RO. В частности, он не подвержен атакам на функции конструкции Narrow Pipe. Нерешен-ными вопросами остаются оптимизация данных алгоритмов выявления оптимального количества раундов хэш-функции.

Выводы

Анализ современных требований показывает, что современные конструкции универсальных криптопримитивов (Skein, Keccak) имеют ряд существенных преимуществ перед классическими схемами, основанными на функциях сжатия. Новое поколение криптопримитивов поддерживает различные комбинации исходных параметров без использования дополнительных средств и приложений. Это позволяет добиться универсальности в реализации нового поколения криптопримитивов на различных архитектурах современных процессоров. Очевидной причиной выбора функции Keccak в качестве победителя стала его конструкция, что позволило обосновать универсальность криптопримитива, реализовать доказуемую стойкость к целому классу атак без увеличения сложности реализации.

Список литературы: 1. http://googleonlinesecurity.blogspot.de/2014/09/gradually-sunsetting-sha-1.html. 2. https://technet.microsoft.com/library/security/2880823. 3. *Бабенко Л. К., Сидоров И. Д, Кириллов А. С.* Ускорение вычислений дискретного логарифма с помощью технологии CUDA // Известия ЮФУ. Технические науки. – 2010. – №11. 4. *Красилов А. А.* Использование технологии CUDA для неграфических вычислений на GPU / А.А. Красилов // Информационные технологии и системы 2013 (ИТС 2013) : материалы междунар. науч. конф., БГУИР, Минск, Беларусь, 23 октября 2013 г. 5. *Беспалов Д. В., Булавинцев В. Г., Семенов А. А.* Использование графических ускорителей в решении задач криптоанализа // ПДМ. Приложение. – 2010. – № 3. 6. *Халимов Г.* Стойкий к коллизиям алгоритм Whirlpool / Геннадий Халимов, Евгений Котух // Правове, нормативне та метрологічне забезпечення системи захисту інформації в Україні : наук.-техн. зб. – 2005. – Вип. 10. – С. 159-165. 7. *D. R. L. Brown,* Generic groups, collision resistance, and ECDSA. Available at http://eprint.iacr.org/2002/026/2002 8. http://keccak.noekeon.org/ 9. http://ehash.iaik.tugraz.at/wiki/The_SHA-3_Zoo. 10. *Andreeva E., Mennink B., Preneel B. & Skrobot M.* (2012), Security Analysis and Comparison of the SHA-3 Finalists BLAKE, Grostl, JH, Keccak, and Skein.from Katholieke Universiteit Leuven. 11. *Bertoni G., Daemen J., Peeters M.l. and G.Van Assche* The Keccak SHA-3 submission, 2011. 12. *Morawiecki P., Pieprzyk J.* Parallel authenticated encryption with the duplex construction, Cryptology ePrint Archive: Report 2013/658. 13. *Fleischmann E., Forler C. and M. Gorski*. Classification of the SHA-3 candidates. – Cryptology ePrint Archive, Report 2008/511, 2008. 14. http://hashcat.net/oclhashcat/

Харьковский национальный
университет радиоэлектроники

Поступила в редколлегию 25.04.2015