MCSSHA

Семейство алгоритмов хеширования

Основано на использовании неавтономного регулярного регистра сдвига над Z/256

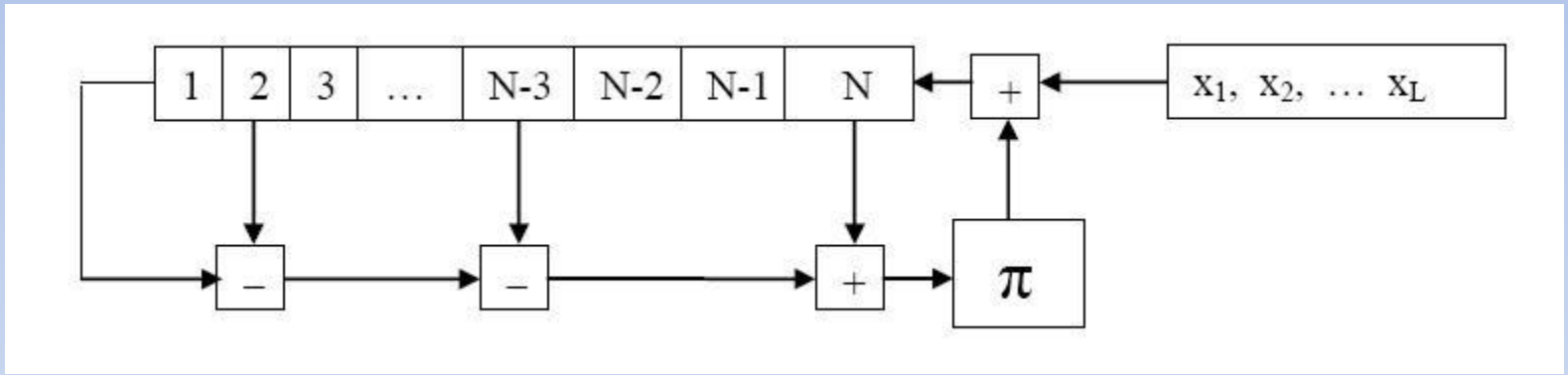
Автор: Масленников М.Е., начальник отдела разработки ПО предприятия НТЦ «Система»
Последнее обновление: 20 марта 2014 г.

ВВЕДЕНИЕ

- Любой алгоритм хеширования из семейства MCSSHA является итерактивной однонаправленной хеш-функцией, результатом выполнения которой является *дайджест* произвольного сообщения, состоящего из набора байт. Хеш функция позволяет проверять *целостность* сообщения, поскольку любое изменение в сообщении приводит к изменению его дайджеста, причем с очень высокой вероятностью два различных сообщения будут иметь различные дайджесты, отличающиеся друг от друга как случайные и равновероятные величины. Это свойство позволяет использовать хеш-функцию в системах электронной подписи, в системах выработки и проверки различных идентификационных кодов, для выработки случайных чисел и в других подобных задачах.

КРИПТОГРАФИЧЕСКАЯ СХЕМА

- Криптосхема MCSSHA, основанная на неавтономном регулярном регистре сдвига (SR) над Z/256, приведена ниже.



где N – длина регистра (в байтах);

π – подстановка из симметрической группы S_{256};

$x_1, x_2, \dots x_L$ – входная последовательность в байтах, т.е. из Z/256.

Элементами SR являются байты, т.е. элементы из Z/256,

Состоянием SR (SR state) является вектор $(y_i, y_{i+1}, \dots, y_{i+N-1})$, в котором любая координата $y_i, y_{i+1}, \dots, y_{i+N-1}$ является байтом, т.е. элементом из Z/256.

Если $(y_i, y_{i+1}, \dots, y_{i+N-1})$ - SR state перед i-ым шагом, то после i-го шага SR state будет $(y_{i+1}, y_{i+2}, \dots, y_{i+N})$, где

$$y_{i+N} = \pi(y_i - y_{i+1} - y_{i+N-4} + y_{i+N-1}) + x_i$$

История разработки MCSSHA

- Первый алгоритм из семейства MCSSHA – MCSSHA-3 был разработан в 2008 году для проводимого американским NIST международного открытого конкурса SHA-3. В ходе этого конкурса независимыми экспертами Jean-Philippe Aumasson (Швейцария) и Mar´ıa Naya-Plasencia (Франция) были найдены некоторые несоответствия MCSSHA-3 требованиям, предъявляемым NIST. Автором были проанализированы причины этих несоответствий и предложена модернизация алгоритма MCSSHA-3 – алгоритм MCSSHA-4. Через некоторое время те же эксперты нашли уже в MCSSHA-4 отклонение от требований NIST, и автором были предложены две модернизации: MCSSHA-5 и MCSSHA-6.
- Во всех методах, применяемых экспертами при атаках на хеш-функции семейства MCSSHA, использовался в том или ином виде «парадокс дней рождения», требующий значительного объема памяти. Вопрос о реальном нахождении возможных коллизий не рассматривался.

История разработки MCSSHA

- Алгоритмы хеширования MCSSHA-7 и MCSSHA-8 были разработаны в 2013 и в 2014 гг. после завершения конкурса на создание нового стандарта хеширования SHA-3. Был учтен опыт предыдущих ошибок, допущенных в MCSSHA-3 – MCSSHA-6 и, при сохранении высокой скорости работы и простоты реализации, найдены кардинальные методы защиты от атак, которым они подвергались при проведении криптоанализа независимыми международными экспертами.
- При построении и анализе алгоритмов хеширования MCSSHA-7 и 8 автор руководствовался теми же требованиями NIST, которые предъявлялись для участников конкурса SHA-3 за одним важным исключением: в MCSSHA-7 и 8 нет требования ограничиться значениями длины дайджеста в 224, 256, 384 или 512 бит, т.е. 28, 32, 48 или 64 байта соответственно. Принцип построения MCSSHA-7 и 8 позволяет вычислять дайджест для любого целого значения длины в байтах от 4 до 64.

Этапы MCSSHA

- Работа любого алгоритма типа MCSSHA осуществляется в три этапа: preprocessing, pre-hash computation и final hash computation. На каждом этапе изменяются состояния SR. Длины SR на различных этапах также могут быть различными.

  Preprocessing: устанавливается начальное состояние SR, не зависящее от хешируемого сообщения.

  The pre-hash computation: выработка pre-final SR-state из начального SR state и хешируемого сообщения.

  The final hash computation: выработка окончательной хеш-функции - message digest – из pre-final SR-state. Байты хешируемого сообщения на вход не подаются

Параметры регистра сдвига

- Пусть N – длина SR в байтах. Параметрами SR являются:
  - **Состояние SR** - N байт: $(y_0, y_1, \ldots, y_{N-1})$;
  - **Точки съема SR** – 4 значения от 0 до N-1: (p_1, p_2, p_3, p_4) ;
  - **Подстановка SR** – группа из 256 байт в которой все величины различные. Взаимно-однозначное преобразование π для байт 0,1,…,255 будем обозначать π(y) – замена байта y по подстановке π.
  - Состояние и точки съема SR меняются при каждом шаге работы алгоритма. Подстановка π остается неизменной.

Шаг регистра сдвига

- Шаг регистра сдвига (SR step) – это преобразование состояния регистра (SR state) и точек съема (SR points) за один шаг работы.

  Пусть:

  $Y=(y_0,y_1, \ldots,y_{N-1})$ – SR state перед шагом;

  $P=(p_1,p_2,p_3,p_4)$ – SR points перед шагом

  p – изменяемая позиция: $p= (p_4+1)(\mathrm{mod}\ N)$;

  x – входной байт для шага.

  Тогда SR state F1(Y,P,x) и SR points F2(P) после шага будут следующими:

$$F1(Y,P,x) = (y_0,y_1, \ldots,y_{p-1},z,y_{p+1},\ldots,y_{N-1}) \quad 0 < p < N-1$$
$$F1(Y,P,x) = (z,y_1,y_2, \ldots,y_{N-1}) \quad p = 0$$
$$F1(Y,P,x) = (y_0,y_1, \ldots,y_{N-2},z) \quad p = N-1$$

  где

$$z = \pi(y_{p1} - y_{p2} - y_{p3} + y_{p4}) + x$$

$$F2(P) = ((p_1+1)(\mathrm{mod}\ N), (p_2+1)(\mathrm{mod}\ N), (p_3+1)(\mathrm{mod}\ N), (p_4+1)(\mathrm{mod}\ N))$$

Логарифмическая подстановка π

- Для создания хороших криптографических свойств преобразования состояний SR, подстановка должна обеспечивать «лавинообразный эффект» размножения различий в состояниях. Такой «лавинообразный эффект» описывается так называемой матрицей переходных вероятностей ненулевых биграмм P(π) подстановки π размера 255x255. В этой матрице на пересечении i-ой строки и j-го столбца находится элемент p_{ij} – число решений системы:

$$x - y = i$$
$$\pi(x) - \pi(y) = j$$

в Z/256 для всех i,j ≠0. «Лавинообразный эффект» будет тем выше, чем меньше нулей содержит матрица P(π). Относительно легко доказать, что число нулей в этой матрице не может быть меньше, чем 253.

Известен пример подстановок с минимально возможным количеством нулей в матрице P(π) – это логарифмические подстановки, т.е.

$$\pi(x) = \log_\theta(\theta^{x+r} \oplus \rho), \quad \text{при } \theta^{x+r} \oplus \rho \neq 0,$$
$$\pi(x) = \log_\theta \rho, \quad \text{при } \theta^{x+r} \oplus \rho = 0$$

Здесь:

θ – примитивный элемент поля GF(257),

ρ – ненулевой элемент поля GF(257),

r – произвольный элемент кольца Z/256.

Здесь используются два типа операций сложения:

“+”	сложение в Z/256,
“⊕“	сложение в GF(257).

- Подстановка π во всех алгоритмах типа MCSSHA является логарифмической подстановкой, в которой θ = 3, ρ = 1, r=0.

Длина регистра сдвига

- В алгоритмах MCSSHA-7 и 8 длины SR для этапов preprocessing и pre-hash computation совпадают.

Длина хеш-функции в байтах	Длина регистра в байтах на этапах preprocessing и pre-hash
4	8
От 5 до 8	16
От 9 до 16	32
От 17 до 32	64
От 33 до 64	128

- В алгоритме MCSSHA-8 длина SR на этапе final hash computation совпадает с длиной хеш-функции.

Начальные параметры регистра сдвига

- Если N – длина SR, то начальными параметрами SR будут:

Начальное заполнение SR:

0 1 2 … N-1

Начальные точки съема SR:

$$P_1 = 0$$
$$P_2 = 1$$
$$P_3 = N-4$$
$$P_4 = N-1$$

PRE-HASH COMPUTATION

- Pre-hash computation готовит SR state, которое зависит от всех знаков текста, за исключением оставшихся бит, в случае, когда длина текста в битах не кратна 8. Для каждого байта m_i из сообщения M pre-hash computation выполняет два типа шагов:

  Тип 1: SR step со входным байтом m_i.

  Тип 2: SR step со входным байтом 0 (задержка).

  Любой алгоритм типа MCSSHA во время pre-hash computation выполняет один шаг типа 1 (со входным байтом сообщения) и несколько шагов типа 2 (задержка) Значение задержки может быть установлено в качестве параметра алгоритма.

  При вычислении дайджеста сообщения величина задержки должна быть не менее 2.

  В MCSSHA-7 и 8 величина задержки равна 3.

  Начальными параметрами для pre-hash computation являются начальные параметры SR.

FINAL HASH COMPUTATION для MCSSHA-7

- Final hash computation (FHC) отличаются у всех алгоритмов семейства MCSSHA.
- Для MCSSHA-7 FHC состоит из двух последовательно выполняемых процедур:

- вычисление значения с использованием SR той же длины, что и на этапе phe-hash computation;

- вычисление значения с использованием SR длины, равной длине хеш-функции;

Обе процедуры начинают работу с начальными параметрами SR (SR state и SR points), а на вход SR подаются без задержек результаты предыдущих вычислений – сначала прямое, а затем инвертированное SR state. SR state после последней процедуры является общим дайджестом всего хешируемого сообщения.

FINAL HASH COMPUTATION для MCSSHA-8

- Для MCSSHA-8 FHC состоит из трех последовательно выполняемых процедур:

- вычисление значения с использованием SR длины, равной длине хеш-функции, из первой половины SR state после phe-hash computation;

- вычисление значения с использованием SR длины, равной длине хеш-функции, из второй половины SR state после phe-hash computation;

- покоординатное сложение по модулю 2 вычисленных векторов состояний SR. Полученный результат является общим дайджестом всего хешируемого сообщения.

Особенности алгоритмов хеширования MCSSHA

- Все алгоритмы семейства MCSSHA построены по ***поточному*** принципу обработки хешируемой информации, байт за байтом. Следовательно, в отличие от большинства алгоритмов, построенных по ***блочному*** принципу обработки хешируемой информации, блок за блоком, в них не требуется такой процедуры, как ***padding***, т.е. дополнения последнего неполного блока до полного.
- Все операции в любом алгоритме из семейства MCSSHA осуществляются с байтами. Операции для 16, 32 и 64 – разрядной арифметики не используются. Следовательно, MCSSHA может быть реализован практически с помощью любого процессора, даже 8-разрядного.

Память, требуемая для реализации MCSSHA

- Объем памяти, требуемый для реализации алгоритмов из семейства MCSSHA, складывается из памяти, требуемой для реализации самого алгоритма хеширования и его параметров. Для параметров (подстановки и точек съема) требуется 260 байт, для состояния SR еще не более 128 байт.
- Объем программного кода (статическая библиотека на языке C++ для 64-разрядной версии Windows) для MCSSHA-8 составляет 4798 байт.

Оценка скорости работы MCSSHA на eBACS

- Оценка скорости работы алгоритмов семейства MCSSHA на eBACS (http://bench.cr.yp.to/results-hash.html)

armeabi (v7-A, Cortex A8); 2012 TI Sitara XAM3359AZCZ100; 1 x 1000MHz;

Cycles/byte for long messages			
Quartile	*Median*	*Quartile*	*Hash*
58.57	58.62	58.75	Keccakc1024
60.91	61.04	61.26	mcssha6
63.47	63.59	63.83	keccakc768
64.97	65.04	65.23	mcssha5
64.98	65.04	65.26	mcssha4

Оценка скорости работы MCSSHA на eBACS

- Оценка скорости работы алгоритмов семейства MCSSHA на eBACS (http://bench.cr.yp.to/results-hash.html)

armeabi (v7-A, Cortex A8); 2012 TI Sitara XAM3359AZCZ100; 1 x 1000MHz;

Cycles/byte for 4096 bytes			
Quartile	*Median*	*Quartile*	*Hash*
58.95	58.95	59.01	keccakc1024
62.10	62.10	62.21	cubehash832
62.69	62.70	62.80	mcssha6
64.77	64.77	64.88	keccakc768
66.29	66.30	66.40	mcssha4
66.98	66.99	67.08	mcssha5

Оценка скорости работы MCSSHA на eBACS

- Оценка скорости работы алгоритмов семейства MCSSHA на eBACS (http://bench.cr.yp.to/results-hash.html)

armeabi (v7-A, Cortex A8); 2012 TI Sitara XAM3359AZCZ100; 1 x 1000MHz;

Cycles/byte for 1536 bytes			
Quartile	*Median*	*Quartile*	*Hash*
61.02	61.03	61.31	keccakc1024
61.35	61.39	61.95	fugue2
65.20	65.20	65.21	keccakc768
65.45	65.47	65.51	mcssha6
68.39	68.40	68.46	mcssha4
70.25	70.25	70.27	mcssha5

Внутренние тесты по сравнению скоростей различных алгоритмов хеширования

- Компьютер: Intel Core i7 – 3537U CPU 2,0 GHz 2,50 GHz. Программа тестирования MCSSHA_Tests
- *Таблица 1. Результаты измерения скорости работы различных алгоритмов хеширования.*

Алгоритм	Длина дайджеста в байтах	Количество циклов	Длина текста в цикле в байтах	Время выполнения теста в сек.
Keccak	32	1	1000000	0,088
MCSSHA-7	32	1	1000000	0,131
MCSSHA-8	32	1	1000000	0,125
GOST 3411	32	1	1000000	0,174
GOST 2012 SLOW	32	1	1000000	31,387
GOST 2012 FAST	32	1	1000000	0,88
SHA-2 (OpenSSL)	32	1	1000000	0,078
Keccak	32	1000	1000	0,093
MCSSHA-7	32	1000	1000	0,154
MCSSHA-8	32	1000	1000	0,146
GOST 3411	32	1000	1000	0,188
GOST 2012 SLOW	32	1000	1000	36,137
GOST 2012 FAST	32	1000	1000	0,981
SHA-2 (OpenSSL)	32	1000	1000	0,078

Внутренние тесты по сравнению скоростей различных алгоритмов хеширования

- Компьютер: Intel Core i7 – 3537U CPU 2,0 GHz 2,50 GHz. Программа тестирования MCSSHA_Tests
- *Таблица 1. Результаты измерения скорости работы различных алгоритмов хеширования.*

Алгоритм	Длина дайджеста в байтах	Количество циклов	Длина текста в цикле в байтах	Время выполнения теста в сек.
Keccak	32	1000000	1	10,5
MCSSHA-7	32	1000000	1	11,216
MCSSHA-8	32	1000000	1	4,9
GOST 3411	32	1000000	1	23,01
GOST 2012 SLOW	32	1000000	1	∞
GOST 2012 FAST	32	1000000	1	153,598
SHA-2 (OpenSSL)	32	1000000	1	3,79
Keccak	64	1	1000000	0,172
MCSSHA-7	64	1	1000000	0,125
MCSSHA-8	64	1	1000000	0,094
GOST 3411	64	1	1000000	0,156
GOST 2012 SLOW	64	1	1000000	30,997
GOST 2012 FAST	64	1	1000000	0,826
SHA-2 (OpenSSL)	64	1	1000000	0,078

Внутренние тесты по сравнению скоростей различных алгоритмов хеширования

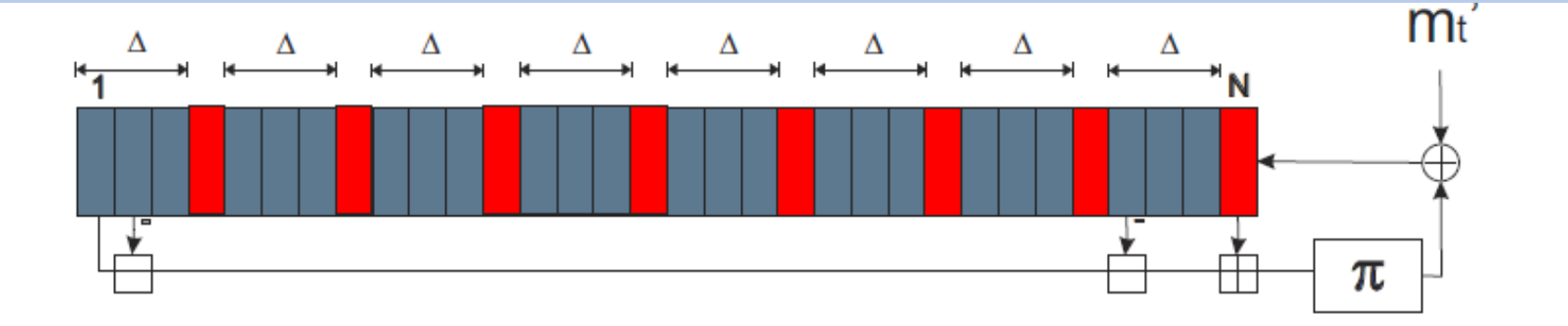
- Компьютер: Intel Core i7 – 3537U CPU 2,0 GHz 2,50 GHz. Программа тестирования MCSSHA_Tests
- *Таблица 1. Результаты измерения скорости работы различных алгоритмов хеширования.*

Алгоритм	Длина дайджеста в байтах	Количество циклов	Длина текста в цикле в байтах	Время выполнения теста в сек.
Keccak	64	1000	1000	0,156
MCSSHA-7	64	1000	1000	0,156
MCSSHA-8	64	1000	1000	0,141
GOST 2012 SLOW	64	1000	1000	35,069
GOST 2012 FAST	64	1000	1000	0,936
SHA-2 (OpenSSL)	64	1000	1000	0,109
Keccak	64	1000000	1	10,545
MCSSHA-7	64	1000000	1	22,23
MCSSHA-8	64	1000000	1	8,69
GOST 2012 SLOW	64	1000000	1	∞
GOST 2012 FAST	64	1000000	1	151,976
SHA-2 (OpenSSL)	64	1000000	1	11,482

Криптографический анализ

Как видно из описания, неавтономный регулярный регистр сдвига SR используется на этапах предварительного и окончательного вычисления дайджеста, причем это могут быть регистры, различающиеся по своим параметрам.

В первом варианте (MCSSHA-3) длины обоих регистров были одинаковыми и равнялись длине дайджеста. Этот факт использовали Jean-Philippe Aumasson and Mar´ıa Naya-Plasencia в своей работе «Cryptanalysis of the MCSSHA Hash Functions».



С помощью birthday attack они подбирали сообщения, у которых совпадают значения на местах, закрашенных серым цветом, а затем, пользуясь тем, что на «красных» местах находятся значения, линейно зависящие от знаков входного текста, однозначно подбирали такие, при которых все значения регистра будут идентичными. Трудоемкость нахождения коллизии равнялась трудоемкости birthday attack для случайных векторов длины ¾N, что не соответствовало требованиям NIST, хотя и не открывало возможностей для дальнейшего снижения стойкости и не могло рассматриваться как практически значимая опасность.

Криптографический анализ

Защита от подобного метода была найдена быстро и эффективно, что признали сами авторы атаки. Эта защита заключалась во внесении избыточности на этапе предварительного вычисления дайджеста путем увеличения длины регистра в два раза. На скорости работы алгоритма это практически не отразилось, поскольку в программной реализации, естественно, никаких сдвигов всей памяти не было, а смещались только 4 указателя на точки съема, причем само смещение приводилось по модулю N. Увеличение длины регистра позволило также уменьшить длину задержки до 2 и даже увеличить при этом общую скорость работы алгоритма. Соответствующий вариант алгоритма получил наименование MCSSHA-4.

При атаках на MCSSHA-4 те же оппоненты сосредоточились на этапе заключительного вычисления дайджеста, когда на вход регистра подается два раза заполнение регистра двойной длины 2N от этапа предварительного вычисления. Длина регистра на заключительном этапе в MCSSHA-4 осталась равной длине дайджеста, а начальное заполнение – 0,1,2,…,N-1.

Суть метода их атаки приводится ниже на примере N=256 (32 байта).

Одним из требований к идеальной хеш-функции является то, чтобы трудоемкость построения сообщения с фиксированным значением хеш-функции составляла в данном случае примерно 2^{32*8} операций.

На первом этапе (pre-hash computation) используется регистр длиной 64 байта, а на втором (final hash computation) – 32 байта. Займемся повнимательнее вторым этапом. В нем на регистр длины 32 два раза подается одно и то же входное слово длины 64, при этом начальное заполнение регистра R0 = (0, 1, 2,…,31). Попробуем построить сообщение, которое на первом этапе дает конечное заполнение $y_0,y_1,\ldots,y_{63}$ такое, при котором на втором этапе R0 переходит само в себя. Методика простая: для произвольных случайных $y_0,y_1,\ldots,y_{31}$ вычисляем состояние R1 регистра и для него однозначно находим $y_{32}',y_{33}',\ldots,y_{63}'$, переводящие его в R0. Дергаем случайные сообщения, получаем какие-то y_0, $y_1,\ldots,y_{31}$, вычисляем для них $y_{32}',y_{33}',\ldots,y_{63}'$ и пытаемся дописать сообщение так, чтобы последующие 32 байта совпали с $y_{32}',y_{33}',\ldots,y_{63}'$. Из этих 32 байт 22 – случайные, приходящиеся на «дырки», а 10 – на символы текста, которые однозначно подбираем. Если удалось подобрать, то значение хеш-функции такого сообщения будет совпадать с R0. Трудоемкость – 2^{22*8}, не выполнено условие «preimage resistance of approximately *n* bits».

При построении последних версий – алгоритмов MCSSHA-7 и 8 – этот способ атаки был учтен и этап final hash computation переделан с учетом защиты от подобных методов атаки.

Криптографический анализ. Процесс «склеивания» состояний SR.

Для оценки криптографических свойств алгоритмов семейства MCSSHA попробуем на примере описать процесс «склеивания» двух различных состояний SR и трудоемкость нахождения знаков входного сообщения, обеспечивающих такое «склеивание».

Итак, предположим, что N = 64, задержка 3, и в некоторый такт t произошло «склеивание» состояний X и Y, т.е. $(y_t,y_{t+1},\ldots,y_{t+N-1}) = (z_t,z_{t+1},\ldots,z_{t+N-1})$. Рассмотрим шаги, предшествующие «склеиванию».

Шаг до склеивания	SR states	Следующий шаг	Условия	Вероятность
1	$(y_{t-1},y_t,\ldots,y_{t+62})$ $(z_{t-1},z_t,\ldots,z_{t+62})$	$y_{t+63}=\pi(y_{t-1}-y_t-y_{t+59}+y_{t+62})+m_1(j)$ $z_{t+63}=\pi(z_{t-1}-z_t-z_{t+59}+z_{t+62})+m_2(j)$	Отсутствуют	1
2	$(y_{t-2},y_{t-1},y_t,\ldots,y_{t+61})$ $(z_{t-2},z_{t-1},z_t,\ldots,z_{t+61})$	$y_{t+62}=\pi(y_{t-2}-y_{t-1}-y_{t+58}+y_{t+61})$ $z_{t+62}=\pi(z_{t-2}-z_{t-1}-z_{t+58}+z_{t+61})$	$y_{t-2}-y_{t-1}=z_{t-2}-z_{t-1}$	1 (см. шаг 65)
3	$(y_{t-3},y_{t-2},y_{t-1},y_t,\ldots,y_{t+60})$ $(z_{t-3},z_{t-2},z_{t-1},z_t,\ldots,z_{t+60})$	$y_{t+61}=\pi(y_{t-3}-y_{t-2}-y_{t+57}+y_{t+60})$ $z_{t+61}=\pi(z_{t-3}-z_{t-2}-z_{t+57}+z_{t+60})$	$y_{t-3}-y_{t-2}=z_{t-3}-z_{t-2}$	2^{-8}
4	$(y_{t-4},\ldots,y_{t-1},y_t,\ldots,y_{t+59})$ $(z_{t-4},\ldots,z_{t-1},z_t,\ldots,z_{t+59})$	$y_{t+60}=\pi(y_{t-4}-y_{t-3}-y_{t+56}+y_{t+59})$ $z_{t+60}=\pi(z_{t-4}-z_{t-3}-z_{t+56}+z_{t+59})$	$y_{t-4}-y_{t-3}=z_{t-4}-z_{t-3}$	2^{-8}
5	$(y_{t-5},\ldots,y_{t-1},y_t,\ldots,y_{t+58})$ $(z_{t-5},\ldots,z_{t-1},z_t,\ldots,z_{t+58})$	$y_{t+59}=\pi(y_{t-5}-y_{t-4}-y_{t+55}+y_{t+58})+m_1(j-1)$ $z_{t+59}=\pi(z_{t-5}-z_{t-4}-z_{t+55}+z_{t+58})+m_2(j-1)$	Отсутствуют	1
6	$(y_{t-6},\ldots,y_{t-1},y_t,\ldots,y_{t+57})$ $(z_{t-6},\ldots,z_{t-1},z_t,\ldots,z_{t+57})$	$y_{t+58}=\pi(y_{t-6}-y_{t-5}-y_{t+54}+y_{t+57})$ $z_{t+58}=\pi(z_{t-6}-z_{t-5}-z_{t+54}+z_{t+57})$	$y_{t-6}-y_{t-5}=z_{t-6}-z_{t-5}$	1 (см. шаг 69)

Криптографический анализ. Процесс «склеивания» состояний SR.

Очевидно, что склеивание отдельных знаков можно провести в «активные» такты работы, т.е. когда на вход подаются знаки хешируемого сообщения. Однако некоторые операции, способствующие склеиванию, можно осуществить и в «пассивные» такты, когда на вход SR подаются нули задержки.

Шаг до склеивания	SR states	Следующий шаг	Условия	Вероятность
60	$(y_{t-60},y_{t-59},\dots,y_{t+3})$ $(z_{t-60},z_{t-59},\dots,z_{t+3})$	$Y_{t+4}=\pi(y_{t-60}-y_{t-59}-y_t+y_{t+3})$ $z_{t+4}=\pi(z_{t-60}-z_{t-59}-z_t+z_{t+3})$	$y_{t-60}-y_{t-59}=z_{t-60}-z_{t-59}$	2^{-8}
61	$(y_{t-61},y_{t-60},\dots,y_{t+2})$ $(z_{t-61},z_{t-60},\dots,z_{t+2})$	$Y_{t+3}=\pi(y_{t-61}-y_{t-60}-y_{t-1}+y_{t+2})+m_1(j-15)$ $z_{t+3}=\pi(z_{t-61}-z_{t-60}-z_{t-1}+z_{t+2})+m_2(j-15)$	Отсутствуют	1
62	$(y_{t-62},y_{t-61},\dots,y_{t+1})$ $(z_{t-62},z_{t-61},\dots,z_{t+1})$	$Y_{t+2}=\pi(y_{t-62}-y_{t-61}-y_{t-2}+y_{t+1})$ $z_{t+2}=\pi(z_{t-62}-z_{t-61}-z_{t-2}+z_{t+1})$	$y_{t-62}-y_{t-61}-y_{t-2}=z_{t-62}-z_{t-61}-z_{t-2}$	2^{-8}
63	$(y_{t-63},y_{t-62},\dots,y_t)$ $(z_{t-63},z_{t-62},\dots,z_t)$	$Y_{t+1}=\pi(y_{t-63}-y_{t-62}-y_{t-3}+y_t)$ $z_{t+1}=\pi(z_{t-63}-z_{t-62}-z_{t-3}+z_t)$	$y_{t-63}-y_{t-62}-y_{t-3}=z_{t-63}-z_{t-62}-z_{t-3}$	2^{-8}
64	$(y_{t-64},y_{t-63},\dots,y_{t-1})$ $(z_{t-64},z_{t-63},\dots,z_{t-1})$	$Y_t=\pi(y_{t-64}-y_{t-63}-y_{t-4}+y_{t-1})$ $z_t=\pi(z_{t-64}-z_{t-63}-z_{t-4}+z_{t-1})$	$y_{t-64}-y_{t-63}-y_{t-4}+y_{t-1}=$ $z_{t-64}-z_{t-63}-z_{t-4}+z_{t-1}$	2^{-8}
65	$(y_{t-65},y_{t-64},\dots,y_{t-2})$ $(z_{t-65},z_{t-64},\dots,z_{t-2})$	$Y_{t-1}=\pi(y_{t-65}-y_{t-64}-y_{t-5}+y_{t-2})+m_1(j-16)$ $z_{t-1}=\pi(z_{t-65}-z_{t-64}-z_{t-5}+z_{t-2})+m_2(j-16)$	Добиваемся совпадения $y_{t-2}-y_{t-1}=z_{t-2}-z_{t-1}$	1
66	$(y_{t-66},y_{t-65},\dots,y_{t-3})$ $(z_{t-66},z_{t-65},\dots,z_{t-3})$	$Y_{t-2}=\pi(y_{t-66}-y_{t-65}-y_{t-6}+y_{t-3})$ $z_{t-2}=\pi(z_{t-66}-z_{t-65}-z_{t-6}+z_{t-3})$		

Очевидно, что при любых фиксированных $(y_{t-65},y_{t-64},\dots,y_{t-2})$, $(z_{t-65},z_{t-64},\dots,z_{t-2})$ и $m_2(j-16)$ с помощью подбора $m_1(j-16)$ можно получить пару (y_{t-1}, z_{t-1}) такую, что $y_{t-2}-y_{t-1}=z_{t-2}-z_{t-1}$.Аналогично на шаге 69 подбираем входной знак таким образом, чтобы выполнялось $y_{t-6}-y_{t-5}=z_{t-6}-z_{t-5}$ и.т.д.

ВЫВОДЫ

Алгоритмы хеширования из семейства MCSSHA являются перспективными алгоритмами, обладающими высокой скоростью работы и достаточной криптографической надежностью. Они могут быть реализованы с помощью практически любого процессора, в том числе внутри смарт-карты, при минимальном использовании памяти и ресурсов процессора.

Ссылки и адреса в Интернет

- http://crypto.systema.ru/mcssha.php - хранилище документации по всем алгоритмам семейства MCSSHA
- Jean-Philippe Aumasson and Mar´ıa Naya-Plasencia. Cryptanalysis of the MCSSHA Hash Functions.
- Announcing Request for Candidate Algorithm Nominations for a New Cryptographic Hash Algorithm (SHA–3) Family
- **eBACS: ECRYPT Benchmarking of Cryptographic Systems**
- Ю. Н. Зайко. Криптография глазами физика. Известия *Саратовского университета. 2009. Т. 9.* Сер. Физика, *вып. 2*