Масленников М.Е.

Алгоритм хеширования MCSSHA-7

Оглавление

Введение

MCSSHA-7 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.

Алгоритм хеширования MCSSHA-7 был разработан в 2013 г. после завершения международного открытого конкурса на создание нового стандарта хеширования SHA-3, проводимого американским национальным институтом стандартов и технологий (NIST) в 2007 – 2012 годах. Алгоритму MCSSHA-7 предшествовали несколько аналогичных алгоритмов MCSSHA-3, MCSSHA-4, MCSSHA-5 и MCSSHA-6, которые были выдвинуты на конкурс SHA-3 и в которых в ходе проведения конкурса были найдены некоторые несоответствия их криптографических свойств требованиям NIST – см. [5].

В алгоритме хеширования MCSSHA-7, разработанному почти через 5 лет после первых версий алгоритмов хеширования семейства MCSSHA, был учтен опыт предыдущих ошибок, допущенных в MCSSHA-3 – MCSSHA-6 и, при сохранении высокой скорости работы и простоты реализации, найден кардинальный метод защиты от атак, которым они подвергались при проведении криптоанализа независимыми международными экспертами. Целью настоящей работы является описание и обоснование алгоритма хеширования MCSSHA-7 как одного из перспективных и высокоскоростных алгоритмов, который может найти широкое применение в современных программах и устройствах (например, в смарт картах) для решения различных криптографических задач.

1. История конкурса SHA-3

1.1 Требования NIST к кандидатам конкурса SHA-3

Требования NIST к кандидатам конкурса SHA-3 были опубликованы на сайте NIST в [6] и содержат несколько разделов.

Согласно разделу 1, алгоритм SHA-3 должен прийти на смену алгоритмам SHA-2, которые поддерживают вычисление дайджестов размера 224, 256, 384 и 512 бит и, следовательно, должен поддерживать вычисление дайджестов той же длины. Размер дайджеста 160 бит, реализованный в алгоритме SHA-1, признан недостаточным и замена его алгоритмом SHA-3 не планируется.

Там же приведено требование о том, что алгоритм SHA-3 должен поддерживать те приложения, в которых к настоящему времени уже реализованы алгоритмы семейства SHA-2. Это означает, что процедура вычисления дайджеста должна состоять из трех стадий: инициализация (Init), добавление (Update) и завершение (Final).

Криптографические требования NIST к кандидатам конкурса SHA-3 изложены на сайте NIST в [6] в разделе 4.A Security. Эти требования определяют возможности алгоритма хеширования противостоять различным атакам на него.

Интуитивно ясно, что под атаками на алгоритм хеширования понимаются попытки атакующего криптоаналитика (attacker) построить *коллизию*, т.е. найти пару сообщений с одинаковыми дайджестами. В требованиях NIST специально оговариваются возможности, которыми обладает криптоаналитик:

«If a construct is specified for the use of the candidate algorithm in a randomized hashing scheme, the construct must, with overwhelming probability, provide n *bits of security against the following attack: The attacker chooses a message,* $M1$*. The specified construct is then used on* $M1$*with a randomization value* $r1$ *that has been randomly chosen without the attacker's control after the attacker has supplied* $M1$*. Given* $r1$*, the attacker then attempts to find a second message* $M2$ *and randomization value* $r2$ *that yield the same randomized hash value.»*

Таким образом, схема атаки на алгоритм хеширования выглядит следующим образом:

- криптоаналитик выбирает некоторое произвольное сообщение M1;

- алгоритм хеширования строит из M1 дайджест r1 без контроля со стороны криптоаналитика, после того, как он выбрал M1;

- имея r1, криптоаналитик пытается найти другое сообщение M2 и его дайджест r2, который бы совпадал с r1.

Основными характеристиками любого метода атаки на алгоритм хеширования являются его трудоемкость и объем используемой для атаки памяти.

В подразделе iii раздела 4.A требований NIST приведено следующее:

«iii. Additional Security Requirements of the Hash Functions

In addition to the specific requirements mentioned above, NIST expects the SHA–3 algorithm of message digest size n to meet the following security requirements at a minimum.

These requirements are believed to be satisfiable by fairly standard hash algorithm constructions; any result that shows that the candidate algorithm does not meet these requirements will be considered to be a serious attack.

Collision resistance of approximately n/2 bits,

Preimage resistance of approximately n bits,

Second-preimage resistance of approximately n-k bits for any message shorter than 2k bits,

Resistance to length-extension attacks, and

Any m-bit hash function specified by taking a fixed subset of the candidate function's output bits is expected to meet the above requirements with m replacing n. (Note that an attacker can choose the m-bit subset specifically to allow a limited number of precomputed message digests to collide, but once the subset has been chosen, finding additional violations of the above properties is expected to be as hard as described above.)

Increasing the second preimage resistance property and resistance against other attacks, such as multicollision attacks, will be viewed positively by NIST; however, this could also have performance implications.

Submitters should be prepared to argue for their overall security/performance trade-offs.»

Определение таких понятий, как *collision resistance, preimage resistance и second-preimage resistance* содержится в главе 9 [7]:

collision resistant hash functions (CRHFs): for these, finding any two inputs having the same hash-value is difficult (нахождение двух сообщений с одинаковыми дайджестами является трудоемкой задачей).

preimage resistance—for essentially all pre-specified outputs, it is computationally infeasible to find any input which hashes to that output, i.e., to find any preimage x_0 such that $h(x_0) = y$ when given any y for which a corresponding input is not known.
(практически невозможно за реальное время найти сообщение с тем же дайджестом, что и у неизвестного другого сообщения).

2nd-preimage resistance—it is computationally infeasible to find any second input which has the same output as any specified input, i.e., given x, to find a 2nd-preimage $x' = x$ such that $h(x) = h(x')$.
(практически невозможно за реальное время найти второе сообщение с тем же дайджестом, что и у известного сообщения)

Известен универсальный метод атаки хеш-функций (Algorithm-independent attacks) – это метод, использующий *парадокс дней рождения (Birthday attack)* –см. 9.7.1 [7]. Суть этого метода в том, что при случайном и равновероятном выборе хешируемых сообщений для нахождения первой пары сообщений с одинаковыми дайджестами длины n бит потребуется перебрать в среднем $1{,}25 * 2^{n/2}$ различных сообщений.

Легко видеть, что упомянутые выше требования NIST означают, что для алгоритма – кандидата конкурса SHA-3 не должно быть найдено никаких алгоритмов получения коллизий с трудоемкостью ниже, чем трудоемкость метода birthday attack.

Birthday attack при анализе функций хеширования является в определенном смысле аналогом метода тотального опробования ключей при анализе алгоритмов шифрования. Однако в подобном сравнении есть одно отличие: в методе тотального опробования ключей при анализе алгоритмов шифрования практически не используется память, в то время как для применения birthday attack необходим некоторый объем памяти для хранения перебираемых вариантов

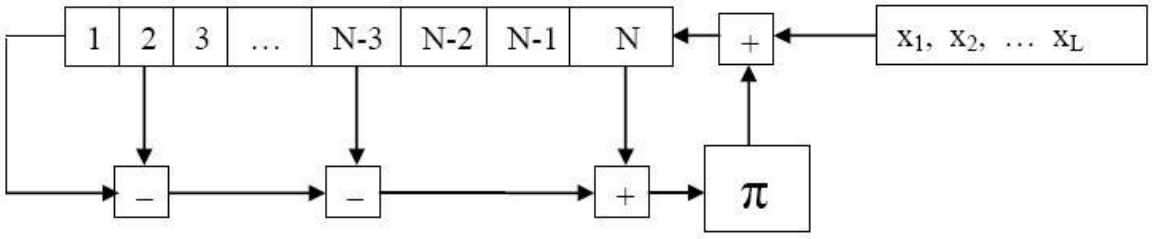
дайджестов с тем, чтобы после набора необходимого количества дайджестов отыскать среди них совпадающую пару.

В классическом варианте birthday attack для нахождения коллизии, т.е. совпадающих дайджестов, необходима память порядка 2^n для хранения сообщений длиной n бит. В [7] предложена модернизация классического варианта birthday attack, минимизирующая используемую память (memoryless variation of birthday attack) и, по-видимому, по этой причине в приведенных выше требованиях NIST отсутствуют какие-либо ограничения на объем памяти, используемой при поиске коллизий.

Ниже приводится история участия алгоритмов хеширования из семейства MCSSHA в конкурсе SHA-3.

1.2 Общий принцип построения алгоритмов типа MCSSHA

В основе построения всех алгоритмов типа MCSSHA лежит использование неавтономного регулярного регистра сдвига (SR – Shift Registry) над кольцом Z/256 с использованием операций сложения и вычитания в этом кольце, а также некоторой подстановки π из симметрической группы S_{256}.



Всюду далее под *тактом* работы SR будем понимать преобразование состояния SR $(y_1,y_2,...,y_N)$ в состояние $(y_2,y_3,...,y_N,y_{N+1})$, где

$$y_{N+1} = \pi(y_1 - y_2 - y_{N-3} + y_N) + x \qquad (1.1)$$

x – элемент Z/256, подаваемый на вход SR.

Построение дайджеста некоторого сообщения M осуществляется в три этапа.

Этап 1. Init, т.е. подготовка начального заполнения регистра сдвига, его точек съема и подстановки.

Этап 2. Update, предварительное вычисление дайджеста.

Этап 3. Final, окончательное вычисление дайджеста.

Отметим некоторые принципиальные моменты в механизме построения всех алгоритмов семейства MCSSHA.

1. Хешируемое сообщение M побайтно подается на вход SR не в каждый такт его работы, а через определенное число тактов, именуемое *задержкой* алгоритма MCSSHA. В такты задержки на вход SR подаются нулевые значения. Благодаря наличию задержки изменение одного байта сообщения на входе вызывает лавинообразное нарастание изменений в состоянии SR, т.е. состояния расходятся. Свести («склеить») однажды разошедшиеся состояния, т.е. подобрать два различных сообщения, для которых в конце состояния SR будут совпадать, можно только с помощью вероятностной модели и перебора значительного числа сообщений.
2. Процедура подачи на вход SR хешируемого сообщения осуществляется только на этапе предварительного вычисления дайджеста. На этапе окончательного вычисления дайджеста на вход SR подается только состояние SR после предварительного этапа. Этап окончательного вычисления дайджеста необходим прежде всего для того, чтобы по окончательному дайджесту нельзя было бы получить какую-то информацию о хешируемом сообщении. Если, к примеру, выдавать заполнение SR после

предварительного этапа в качестве окончательного дайджеста, то используя регулярность преобразования (1.1), при известных начальных знаках сообщения можно легко вычислить его последние знаки.

3. Во всех алгоритмах семейства MCSSHA в качестве подстановки π используется логарифмическая подстановка, обладающая минимально возможным числом нулевых элементов в матрице частот встречаемости разностей биграмм – см [1]. Это дает основания утверждать, что процедура поиска сходимости двух разошедшихся на этапе предварительного вычисления дайджеста состояний SR будет сведена к случайному и равновероятному перебору входных значений сообщения.

Таким образом, предложенный выше метод MCSSHA построения дайджеста обладает следующими особенностями.

1. MCSSHA позволяет получать дайджест сообщения *без разбиения сообщения на блоки и дополнения последнего блока до полного (padding).* Проводя некоторую условную аналогию с алгоритмами шифрования, можно сказать, что MCSSHA построен по принципу гаммирования, в то время как действующие в настоящее время алгоритмы семейств SHA-1 и SHA-2 построены по блочному принципу (см. [8]).
2. Принцип построения алгоритмов MCSSHA позволяет в одной реализации легко обобщить эти алгоритмы для любого значения длины дайджеста, не связывая реализацию условиями кратности длины дайджеста размеру блока.
3. MCSSHA использует только операции с байтами и, в отличии от действующих алгоритмов семейств SHA-1 и SHA-2, не использует операции с 32 или 64 битовыми словами – см. [8]. Это дает возможность легко реализовывать MCSSHA на процессорах практически любой архитектуры, в том числе на 8 и 16 разрядных процессорах.
4. Алгоритмы MCSSHA обладают внутренним механизмом регулирования соотношения «скорость/стойкость», в роли регулятора выступает упоминавшийся выше параметр задержки. Максимальная скорость достигается при нулевой задержке, однако при этом стойкость также можно считать нулевой, поскольку склеивание разошедшихся состояний регистра осуществляется без всякого перебора за счет решения линейных уравнений, возникающих из (1.1). О сложности склеивания при ненулевых значениях параметра задержки речь пойдет ниже.

Из приведенных выше рассуждений можно обрисовать примерную картину возможных атак на алгоритм MCSSHA.

1. Атаки, ставящие своей целью нахождение коллизии на этапе предварительного вычисления дайджеста, т.е. получение совпадающих состояний SR в ходе этапа 2. Такие атаки могут быть осуществлены либо с помощью birthday attack без использования специфических особенностей функции обратной связи регистра, либо, используя уравнения (1.1), за счет специального подбора очередных значений входных байт (*capture attack*).

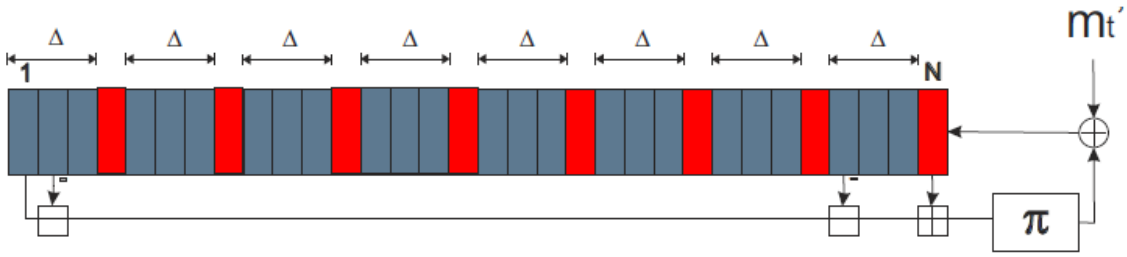
2. Атаки, не дающие коллизий на этапе 2, но приводящие к коллизиям на этапе 3.

Возможность проведения подобных атак для различных версий алгоритма MCSSHA рассматривается в настоящей работе.

1.3 Анализ причин предыдущих успешных атак на алгоритмы семейства MCSSHA

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. Общий вывод по итогам анализа атак состоит в том, что на каждом из этих этапов длина регистра должна быть больше длины дайджеста для того, чтобы обеспечить ***избыточность***, делающую атаки типа birthday attack неэффективными.

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 [5].



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.

Защита от подобного метода была найдена быстро и эффективно, что признали сами авторы атаки в [5]. 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 в [5] 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-image 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».

К сожалению, в последующих версиях MCSSHA-5 и MCSSHA-6 не была устранена главная причина, приводящая к эффективности подобных атак, а именно: отсутствие избыточности на этапе заключительного вычисления дайджеста, а все попытки защититься от подобной атаки сводились только к модификации способа подачи заполнения регистра после первого этапа на регистр второго этапа. Действительно, если, к примеру, при N=256 (32 байта) длину второго регистра также сделать равной 64, то описанный выше метод перестает работать. Пусть начальное заполнение на втором этапе - R0 = (0, 1, 2,…,31,32,…,63) . 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. В этом случае, по аналогии с предыдущим, для произвольных случайных $y_0,y_1,\ldots,y_{31}$ вычисляем состояние R1 регистра длиной 64 байта. Однако при длине второго регистра 64 байта однозначно найти 32 байта $y_{32}',y_{33}',\ldots,y_{63}'$, переводящие его в R0, можно только с вероятностью $2^{-8*32} = 2^{-256}$. Таким образом, трудоемкость опробования возрастает на 2^{256} и метод становится неэффективным.

2. Описание предлагаемого к реализации алгоритма хеширования MCSSHA-7

При построении и анализе алгоритма хеширования MCSSHA-7 мы будем руководствоваться упомянутыми выше требованиями NIST за одним важным исключением: в MCSSHA-7 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 позволяет вычислять дайджест для любого целого значения длины в байтах от 4 до 64. Это позволит использовать MCSSHA-7 для вычисления имитоприставки, а также для вычисления банковских кодов подтверждения достоверности (КПД) в случае, когда банковские технологии предусматривают ограничения на длину КПД.

2.1 Определения

2.1.1 Словарь используемых терминов и обозначений

Bit	0 или 1.
Byte	Группа из 8 бит.
Дайджест, хеш-функция	Результат работы алгоритма MCSSHA-7.
Hash bit length (h)	Длина (в битах) дайджеста сообщения.
Hash byte length (H)	Длина (в байтах) дайджеста сообщения.
Shift Registry length (N)	Целое число.
Shift Registry state (SR state)	Группа из N байт.
Preprocessing	Начальный этап работы алгоритма MCSSHA-7

Pre-hash computation	Этап работы алгоритма MCSSHA-7, обрабатывающий байты сообщения
Final hash computation	Завершающий этап, на котором обрабатываются результаты Pre-hash computation, без поступления новых байт сообщения, за исключением message remain bits
Initial SR state	SR state перед pre-hash computation.
Shift Registry point (SR point)	Число от 0 до N-1.
SR points	Группа из четырех SR points
Initial SR points	SR points перед pre-hash computation.
Shift Registry Substitution	Группа из 256 различных байт.
Shift Registry step (SR step)	Преобразование SR state за один такт.
Input byte for SR step	Входной байт, который использует SR step.
Сообщение (Message)	Группа бит.
Message length in bits	Число бит в сообщении.
Message length in bytes	Число полных байт в сообщении.
Message remain bits	Последние биты в сообщении, не включенные в последний байт.

2.2 Параметры алгоритма, символы и терминология

В настоящем документе при описании алгоритма MCSSHA-7 используются следующие обозначения и параметры.

h	Длина хеш-функции в битах
H	Длина хеш-функции в байтах, H = h/8.
M	Хешируемое сообщение.
l	Длина сообщения M, в битах.
L	Длина сообщения M, в байтах, $L = l/8$.
r	Число оставшихся бит в сообщении, т.е. $r = l - 8L$.
m_i	Байт номер i в сообщении M.
$M=m_1,m_2,...,m_L$	Сообщение M как последовательность байт.
π	Подстановка регистра сдвига.
p_1,p_2,p_3,p_4	Набор SR points. Число точек всегда 4, значения точек меняются с каждым тактом.
Δ	Задержка в процессе pre-hash computation, т.е. число SR steps без входного байта при обработке очередного байта сообщения.

Предполагается, что в MCSSHA-7 длина хеш-функции H может принимать любое значение от 4 до 64, а задержка Δ в процессе pre-hash computation всегда фиксирована и равна 3.

2.2.2 Символы

При описании работы MCSSHA-7 используются следующие символы.

+	Сложение по модулю 256.
-	Вычитание по модулю 256.
π(y)	Замена байта *y* по подстановке *π*.
a(mod N)	Приведение значения *a* по модулю *N*.

2.2.3 Константы

Константой алгоритма MCSSHA-7 является подстановка π, нижний ряд которой в шестнадцатиричном представлении следующий:

30	60	67	B5	43	EA	93	25	48	0D	18	6F	28	7A	FE	B6
D5	9C	23	86	52	42	F7	FD	F6	9B	EE	99	91	BC	2A	63
A1	A0	57	3C	39	D2	EC	71	45	CB	41	DC	0B	5B	C2	36
01	55	7D	FB	ED	83	8F	31	C0	4C	08	E3	9D	C1	D3	E9
B8	BD	AE	0F	E7	70	5A	EB	4D	29	F9	A9	3D	26	46	06
D0	50	A5	BE	66	90	F4	20	E4	33	27	E2	AB	EF	68	54
37	6A	DB	BB	D8	7B	69	C4	F2	BF	85	C7	A6	B4	9A	DD
72	34	E8	FC	D6	21	98	96	32	CA	49	B3	F3	97	8E	2F
00	B0	10	1A	77	38	CF	51	BA	1F	22	AC	62	89	76	C3
02	6E	2C	47	3A	5C	1B	56	8A	5D	03	16	74	58	79	09
D7	F5	0A	92	4F	87	CD	DA	8C	C9	9E	3B	12	6B	53	FF
80	B7	F8	D9	F1	5E	AF	E0	05	A4	14	2B	A3	CC	6C	7C
78	AA	95	84	61	A8	CE	13	88	FA	59	4E	B9	C8	4B	24
D1	07	94	2E	DF	B1	17	A2	1D	4A	C6	AD	15	19	35	7F
81	44	0C	9F	75	7E	D4	82	DE	E6	E1	2D	3E	73	11	8B
C5	A7	F0	6D	1C	64	0E	04	40	1E	8D	E5	3F	B2	65	5F

2.2.4 Функции

Пусть $Y=(y_0,y_1, \ldots,y_{N-1})$- SR state, $P=(p_1,p_2,p_3,p_4)$ – SR points, x – байт на входе, p – позиция, на которую будет подаваться входной символ x *перед* SR step.

На каждом шаге используются функции F1(Y,P,x) и F2(P), которые можно определить следующим образом:

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

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

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

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

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

SR state F1(Y,P,x) и SR point F2(P) становятся SR state и SR points *после* SR step.

2.3 Описание этапов работы MCSSHA-7

2.3.1 Preprocessing

Этап preprocessing выполняется перед тем, как приступить к вычислению хеш-функции. На этом этапе устанавливаются начальные значения SR state и SR points следующим образом:

- если N – длина SR на этапе pre-hash computation, то значением SR state является вектор (0,1,…,N-1);
- для SR points p_1, p_2, p_3, p_4:

$p_1 = 0;$

$p_2 = 1;$

$p_3 = N-4;$

$p_4 = N-1.$

Заметим, что длина SR на этапе preprocessing совпадает с длиной SR на этапе pre-hash computation.

Длина SR N связана с длиной хеш-функции H следующим образом:

$$N = 2^{n+1},$$

где n – минимальное целое число, удовлетворяющее условию

$$2^{(n-1)} < H <= 2^n$$

В явном виде значения длины SR в зависимости от длины хеш-функции приведены ниже в таблице 1.

Таблица 1. Длина SR на этапе pre-hash computation

H - длина хеш-функции в байтах	**N – длина SR**
4	8
От 5 до 8	16
От 9 до 16	32
От 17 до 32	64
От 33 до 64	128

2.3.2 Pre-hash computation

Pre-hash computation обрабатывает все полные байты сообщения M. После обработки очередного байта сообщения, выполняются три шага с нулевыми входными байтами, таким образом, общее число шагов на этапе pre-hash computation для сообщения M длиной L байт составляет 4*L. В результате после выполнения pre-hash computation в SR имеется последовательность из N байт, зависящая от всех полных байт сообщения.

2.3.3 Final hash computation

Final hash computation обрабатывает оставшиеся после pre-hash computation биты, не вошедшие в последний полный байт, а затем вырабатывает итоговое значение дайджеста.

2.3.3.1 Построение входной последовательности для SR на этапе final hash computation

Пусть $b_1,b_2,...,b_r$ – remain bits для сообщения M, $a_1,a_2,...,a_n$ – n bits из SR state после pre-hash computation, где n = 8*N. Входная последовательность Z (в битах) будет следующей:

$Z = b_1,b_2,...,b_r, a_1,a_2,...,a_{n-r}$.

Длина в битах входной последовательности в точности равна n бит, в байтах – N байт.

Если remain bits отсутствуют, то

$Z = a_1,a_2,...,a_n$.

Полученную входную последовательность в байтах обозначим как $Z = Z_1,Z_2,...,Z_N$.

2.3.3.2 Построение предварительного значения хеш-функции на этапе final hash computation

В ходе построения предварительного значения хеш-функции на этапе final hash computation используется SR с теми же параметрами, что и на этапе pre-hash computation. Перед подачей на вход этого регистра входной последовательности 2.3.3.1 осуществляется установка начальных параметров SR:

- начальным значением SR state является вектор (0,1,…,N-1);
- начальным значением для SR points (p_1,p_2,p_3,p_4) является вектор (0,1,N-4,N-1).

После этого осуществляются 2N такта работы SR, первые N тактов при входной последовательности $Z_1,Z_2,...,Z_N$, а последующие N тактов – при $Z_N,Z_{N-1},...,Z_1$. Полученное конечное заполнение SR в байтах обозначим $C_1,C_2,...,C_N$. Его длина равна N - длине SR.

2.3.3.3 Построение окончательного значения хеш-функции на этапе final hash computation

Для вычисления окончательного значения хеш-функции используется SR длиной H, который инициализируется следующим образом:

- начальным значением SR state является вектор (0,1,…,H-1);
- начальным значением для SR points (p_1,p_2,p_3,p_4) является вектор (0,1,H-4,H-1) для H >= 6.

В случае H = 5 (p_1,p_2,p_3,p_4) = (0,1,2,4), а такт работы SR (см. соотношение (1.1)) примет вид:

$$y_6 = \pi(y_1 - y_2 - y_3 + y_5) + x \qquad (2.1)$$

В случае H = 4 (p_1,p_2,p_3,p_4) = (0,1,2,3), а такт работы SR (см. соотношение (1.1)) примет вид:

$$y_5 = \pi(y_1 - y_2 - y_3 + y_4) + x \qquad (2.2)$$

Входной последовательностью является последовательность длины 2N: $C_1,C_2,\ldots,C_N,C_N,C_{N-1},\ldots,C_1$.

После этого осуществляются 2N такта работы SR. Полученное конечное заполнение SR является окончательным значением хеш-функции сообщения.

3. Криптографический анализ MCSSHA-7

Для MCSSHA-7 оценим эффективность предложенных в [5] методов, использующих парадокс «дней рождения» (birthday attack).

3.1 Birthday attack на этапе pre-hash computation

Этот метод использовался для алгоритма MCSSHA-3 (см. раздел 3 [5]). Алгоритм MCSSHA-7 отличается от MCSSHA-3 тем, что в MCSSHA-3 на этапе pre-hash computation длина SR равнялась длине дайджеста, а в MCSSHA-7 на том же этапе длина SR по меньшей мере в 2 раза превосходит длину дайджеста. Таким образом, для MCSSHA-3 длина вектора, для которого проводилась birthday attack, равнялась 3/4H, а для MCSSHA-7 аналогичная длина не меньше 2*(3/4)H = 3/2H, что превосходит длину дайджеста и делает метод неэффективным. Здесь H – длина дайджеста.

3.2 Birthday attack на этапе final hash computation

Этот метод использовался для алгоритма MCSSHA-4 (см. раздел 4 [5]), а также, как утверждается в [5], давал снижение трудоемкости pre-image attack для MCSSHA-5 и 6.

Вкратце этот метод уже обсуждался в разделе 1.3, сейчас же остановимся на нем более подробно.

В алгоритмах MCSSHA-4 – 6 длина SR на этапе final hash computation равнялась длине дайджеста H и входная последовательность длины 2H, полученная после pre-hash computation подавалась на SR два раза.

При любом фиксированном векторе Y = ($y_0,y_1,\ldots,y_{H-1}, y_H,y_{H+1},\ldots,y_{2H-1}$) система из H уравнений вида:

$$y_{i+H} = \pi(y_i - y_{(i+1) \bmod H} - y_{(i+H-4) \bmod H} + y_{(i+H-1) \bmod H}) + x_i, \; i = 0,1,\ldots H-1, \qquad (3.1)$$

всегда имеет единственное решение $x_0,x_1,\ldots,x_{H-1}$, и это решение легко вычисляется.

Таким образом, для SR на этапе final hash computation легко подобрать входную последовательность $X = (x_0,x_1,\ldots,x_{H-1})$, переводящую за H тактов начальное заполнение SR само в себя. В частности, можно подобрать последовательность X, переводящую само в себя фиксированное начальное заполнение SR.

Таким образом,

1) Для каждого фиксированного $Y = (y_0,y_1,\ldots,y_{H-1}, y_H,y_{H+1},\ldots,y_{2H-1})$ существует единственный вектор $X = (x_0,x_1,\ldots,x_{H-1})$, являющийся решением (3.1);
2) Для каждого фиксированного $Y = (y_0,y_1,\ldots,y_{H-1}, y_H,y_{H+1},\ldots,y_{2H-1})$ существует ровно 2^{8*H} векторов $X = (x_0,x_1,\ldots,x_{2H-1})$, являющихся решением (3.1);
3) Потенциальную опасность вызывают решения $X = (x_0,x_1,\ldots,x_{2H-1})$ для Y = (0,1,…,H-1,0,1,…,H-1), поскольку в этом случае появляется возможность исключить несколько знаков из текста и получить тот же самый дайджест либо, как это было в MCSSHA-4, построить сообщение с заранее фиксированным дайджестом – значением (0,1,…,H-1). Общее число таких решений равно 2^{8*H}.

Трудоемкость построения решения $X = (x_0,x_1,\ldots,x_{2H-1})$ для Y = (0,1,…,H-1,0,1,…,H-1) была оценена в [5] и приведена выше в разделе 1.3.

В MCSSHA-7 эта потенциальная угроза устранена за счет увеличения в два раза длины SR при построении предварительного значения хеш-функции в final hash computation, т.е. длина SR в этом процессе равна длине SR при pre-hash computation. В этом случае надо строить решение $X = (x_0,x_1,\ldots,x_{N-1})$ для Y = (0,1,…,N-1,0,1,…,N-1), где N >= 2*H. Такое решение определяется однозначно, трудоемкость его определения составляет трудоемкость опробования 2*3/4H байт, что превосходит H.

3.3 Capture attack

Метод «Capture attack» или атаки с помощью захвата, заключается в использовании специфики уравнений функционирования алгоритма типа MCSSHA. Мы пытаемся оценить трудоемкость «склеивания» разошедшихся состояний SR на этапе pre-hash computation.

Предположим, что в некоторый такт t произошло «склеивание» двух различных состояний SR на этапе pre-hash computation, т.е. $(y_t,y_{t+1},\ldots,y_{t+N-1}) = (z_t,z_{t+1},\ldots,z_{t+N-1})$. Попытаемся проследить процесс «склеивания» и выписать необходимые для него соотношения на некоторых предыдущих тактах: t-1, t-2, t-3, и т.д. Заметим, что при данных обозначениях величины y_{t+N-i} и z_{t+N-i} получаются в тактах

$$t - i \text{ , где } i >= 1 \qquad (3.2)$$

В такте t-1 для состояний $(y_{t-1},y_t,\ldots,y_{t+N-2})$ и $(z_{t-1},z_t,\ldots,z_{t+N-2})$ должны быть выполнены следующие соотношения:

$$y_i = z_i \text{ для } i = t,t+1,\ldots t+N-2, \qquad (3.3)$$

$$y_{t-1} \neq z_{t-1} \qquad (3.4)$$

При этом

$$y_{t+N-1} = \pi(y_{t-1} - y_t - y_{t+N-5} + y_{t+N-2}) + m_1(j)$$

$$z_{t+N-1} = \pi(z_{t-1} - z_t - z_{t+N-5} + z_{t+N-2}) + m_2(j),$$

где $m_1(j)$ и $m_2(j)$ – два различных знака подаваемых на вход сообщений. Для «склеивания» состояний SR достаточно подать такие знаки $m_1(j)$ и $m_2(j)$, при которых, в силу (3.3) и (3.4), выполнено:

$$m_2(j) - m_1(j) = \pi(y_{t-1} - y_t - y_{t+N-5} + y_{t+N-2}) - \pi(z_{t-1} - z_t - z_{t+N-5} + z_{t+N-2})$$

Такие значения заведомо существуют с вероятностью 1. Подобный такт (значение t -1, для которого, согласно (3.2), вычисляются y_{t+N-1} и z_{t+N-1}), когда требуемые для «склеивания» соотношения можно подобрать с вероятностью 1, будем назвать критическим.

В тактах t-2, t-3 и t-4 нужно обеспечить совпадение векторов (y_{t+N-2} y_{t+N-3} y_{t+N-4}) и (z_{t+N-2} z_{t+N-3} z_{t+N-4}), где

$$y_{t+N-2} = \pi(y_{t-2} - y_{t-1} - y_{t+N-6} + y_{t+N-3})$$

$$z_{t+N-2} = \pi(z_{t-2} - z_{t-1} - z_{t+N-6} + z_{t+N-3})$$

$$y_{t+N-3} = \pi(y_{t-3} - y_{t-2} - y_{t+N-7} + y_{t+N-4})$$

$$z_{t+N-3} = \pi(z_{t-3} - z_{t-2} - z_{t+N-7} + z_{t+N-4})$$

$$y_{t+N-4} = \pi(y_{t-4} - y_{t-3} - y_{t+N-8} + y_{t+N-5})$$

$$z_{t+N-4} = \pi(z_{t-4} - z_{t-3} - z_{t+N-8} + z_{t+N-5})$$

Здесь и в последующих двух тактах на вход ничего не подается, поскольку они приходятся на задержку. Совпадение будет только в том и только том случае, когда

$$y_{t-2} - y_{t-1} = z_{t-2} - z_{t-1} \qquad (3.5)$$

$$y_{t-3} - y_{t-2} = z_{t-3} - z_{t-2} \qquad (3.6)$$

$$y_{t-4} - y_{t-3} = z_{t-4} - z_{t-3} \qquad (3.7)$$

Поскольку задержка *Δ* равна 3, то критическими тактами будут t – 1 - 4*k, где k >= 0. Согласно (3.2), значения y_{t-2} и z_{t-2} получаются в такт t – 1 - N, т.е. также являются критическими, поскольку N кратно 4 при любой длине дайджеста. За счет свойств логарифмической подстановки, при любых фиксированных y_{t-3} и z_{t-3} с вероятностью, близкой к 1, можно подобрать такие значения входных символов, при которых будет выполнено (3.6). Таким образом, совпадение символов y_{t+N-3} и z_{t+N-3} обеспечивается с вероятностью, близкой к 1, за счет подбора пары входных символов на такте t – 1 - N.

Однако в тактах t – 2 и t - 4, когда нужно обеспечить совпадение знаков y_{t+N-2} с z_{t+N-2} и y_{t+N-4} с z_{t+N-4}

ни одно из значений t-2, t-4 не является критическим и требуемые соотношения (3.5) и (3.7) могут получиться с вероятностью 2^{-8} каждое про случайном и равновероятном выборе сообщений. В данной ситуации будет справедлива случайная и равновероятная модель выбора значений разностей из (3.5) и (3.7) за счет свойств логарифмической подстановки.

Таким образом, при «склеивании» двух разошедшихся состояний SR на этапе pre-hash computation с помощью capture attack мы руководствуемся вероятностной моделью, которая предусматривает опробование по меньшей мере $2^{-8*N/2}$ случайных сообщений, что соответствует требованиям NIST.

4. Оценки эффективности алгоритма хеширования MCSSHA-7

Оценки эффективности алгоритма MCSSHA-7 будем проводить по нескольким параметрам и сравнивать их с наиболее известными мировыми и российскими аналогами алгоритмов хеширования.

4.1 Скорость работы

Для оценки скорости работы различных алгоритмов хеширования была разработана специальная программа MCSSHA_Tests, которая оценивает общее время выполнения типовых процедур хеширования. В ней в начале работы задаются параметры тестирования (длина сообщения и количество циклов хеширования случайных сообщений этой длины), а затем вычисляется общее время выполнения теста. Ниже приводится исходный код реализации алгоритма тестирования для MCSSHA-7, написанный на языке C#.

```
// MCSSHA7 speed
private: System::Void MCSSHA7Speed(System::Object^  sender,
System::Windows::Forms::LinkLabelLinkClickedEventArgs^  e) {
           MCSSHA7_CTX c;
           DWORD dwTextLen;
           char *text = NULL;
           DWORD dwHashBitLen;
        DWORD dwTestsNum;
        DWORD i;
           char h[64];
           String ^Res;
           float TstTime;
        clock_t start,finish;

           ReadParameters(&dwHashBitLen,&dwTextLen,&dwTestsNum);

           resultBox->AppendText("\n");
           resultBox->AppendText("\n##################");
           resultBox->AppendText("\nMCSSHA-7 speed test");
           resultBox->AppendText("\nLanguage: C");
           resultBox->AppendText("\nHashBitLen = " + hashLenCombo->Text);
           resultBox->AppendText("\nNumber of tests = " + numTestsEdit->Text);
           resultBox->AppendText("\nText length = " + textLenEdit->Text);

// Generate random text
        if(!GenerateFixedMessage(&text,dwTextLen))goto stop;
// Text length in bits
           dwTextLen = dwTextLen<<3;
```

```
            start = clock();

          for(i=0;i<dwTestsNum;i++)
            {
// One test cycle
                  MCSSHA7_Init(&c,dwHashBitLen);
                  MCSSHA7_Update(&c,(const BitSequence *)text,dwTextLen);
                  MCSSHA7_Final(&c,(BitSequence *)h);
            }
            finish = clock();

            TstTime = (float)(finish - start)/CLOCKS_PER_SEC;
            Res = "\nTime: " + TstTime.ToString() + " sec.";
            resultBox->AppendText(Res);

stop:
            if(text != NULL)free(text);
             }
```

В Таблице 1 приведены результаты измерения скорости работы MCSSHA-7 и других известных российских и зарубежных алгоритмов хеширования, осуществленные с помощью программы MCSSHA_Tests. Для последнего российского алгоритма хеширования (ГОСТ Р 3411-20 – см. [9]) применяются два варианта реализации: медленный, без использования дополнительной памяти и более быстрый, для которого нужно свыше 14 Кб дополнительной памяти и заполнение ее на предварительном этапе, который не вошел в тестовый цикл. В Таблице 1 они обозначены как GOST2012 Slow и GOST2012 Fast соответственно.

Компьютер: Intel Core i7 – 3537U CPU 2,0 GHz 2,50 GHz

Таблица 1. Результаты измерения скорости работы различных алгоритмов хеширования.

№ теста	Название алгоритма	Длина хеш-функции в битах	Кол-во циклов в тесте	Длина текста в цикле в байтах	Время выполнения теста в сек.
1	Keccak	256	1	8000000	0,109
2	MCSSHA-7	256	1	8000000	0,172
3	GOST 3411	256	1	8000000	0,312
4	GOST2012 Slow	256	1	8000000	49,343
5	GOST2012 Fast	256	1	8000000	1,186
6	SHA-2 (OpenSSL)	256	1	8000000	0,094
7	Keccak	256	1000	10240	0,172
8	MCSSHA-7	256	1000	10240	0,25
9	GOST 3411	256	1000	10240	0,421
10	GOST2012 Slow	256	1000	10240	66,129
11	GOST2012 Fast	256	1000	10240	1,575
12	SHA-2 (OpenSSL)	256	1000	10240	0,14
13	Keccak	256	1000000	8	1,919
14	MCSSHA-7	256	1000000	8	2,496
15	GOST 3411	256	1000000	8	4,851
16	GOST2012 Slow	256	1000000	8	1199,377
17	GOST2012 Fast	256	1000000	8	28,86
18	SHA-2 (OpenSSL)	256	10000000	8	0,749

19	Keccak	512	1	8000000	0,234
20	MCSSHA-7	512	1	8000000	0,171
21	GOST2012 Slow	512	1	8000000	49,966
22	GOST2012 Fast	512	1	8000000	1,232
23	SHA-2 (OpenSSL)	512	1	8000000	0,125
24	Keccak	512	1000	10240	0,297
25	MCSSHA-7	512	1000	10240	0,234
26	GOST2012 Slow	512	1000	10240	65,574
27	GOST2012 Fast	512	1000	10240	1,607
28	SHA-2 (OpenSSL)	512	1000	10240	0,172
29	Keccak	512	1000000	8	1,919
30	MCSSHA-7	512	1000000	8	4,29
31	GOST2012 Slow	512	1000000	8	1192,918
32	GOST2012 Fast	512	1000000	8	28,954
33	SHA-2 (OpenSSL)	512	1000000	8	1,95

Сравним скорость работы MCSSHA-7 с существующими в настоящее время российскими алгоритмами хеширования: ГОСТ Р 34.11 (94 года) и ГОСТ Р 34.11 20 (2012 года). Результаты приведены ниже в Таблице 2.

Таблица 2. Сравнение скорости работы MCSSHA-7 с существующими в настоящее время российскими стандартами хеширования.

№ теста	Название алгоритма	Длина хеш-функции в битах	Кол-во циклов в тесте	Длина текста в цикле в байтах	Отношение времени выполнения к аналогичному времени для MCSSHA-7
1	GOST 3411	256	1	8000000	1,8
2	GOST2012 Slow	256	1	8000000	286,9
3	GOST2012 Fast	256	1	8000000	6,9
4	GOST 3411	256	1000	10240	1,7
5	GOST2012 Slow	256	1000	10240	264,5
6	GOST2012 Fast	256	1000	10240	6,3
7	GOST 3411	256	1000000	8	1,9
8	GOST2012 Slow	256	1000000	8	480,5
9	GOST2012 Fast	256	1000000	8	11,56
10	GOST2012 Slow	512	1	8000000	292,2
11	GOST2012 Fast	512	1	8000000	7,2
12	GOST2012 Slow	512	1000	10240	280,2
13	GOST2012 Fast	512	1000	10240	6,9
14	GOST2012 Slow	512	1000000	8	278
15	GOST2012 Fast	512	1000000	8	6,7

Анализ таблицы 2 позволяет сделать следующие простые выводы относительно того выигрыша по скорости, который можно получить в результате замены существующих в настоящее время российских алгоритмов хеширования на MCSSHA-7:

1) При длине хеш-функции 32 байта – в 1,7 раза и более;
2) При длине хеш-функции 64 байта – в 6,7 раза и более.

В алгоритме MCSSHA-7 для защиты от потенциальных угроз приняты некоторые меры, приводящие к снижению скорости по сравнению с MCSSHA 3 – 6. К таким мерам следует отнести, в первую очередь, появление дополнительных операций на этапе final hash computation, а также выбор значения задержки, равного 3 (вместо 2 как в MCSSHA-4). Поэтому представляет интерес сравнение скорости предыдущих версий MCSSHA с MCSSHA-7. Результаты такого сравнения приведены ниже в таблице 3. Условия проведения испытаний – те же, что и ранее.

Таблица 3. Результаты измерения скорости работы различных версий алгоритмов семейства MCSSHA.

№ теста	Название алгоритма	Длина хеш-функции в битах	Кол-во циклов в тесте	Длина текста в цикле в байтах	Время выполнения теста в сек.
1	MCSSHA-3	256	1	8000000	0,156
2	MCSSHA-4 (delay=2)	256	1	8000000	0,125
3	MCSSHA-4 (delay=3)	256	1	8000000	0,156
4	MCSSHA-5	256	1	8000000	0,171
5	MCSSHA-6	256	1	8000000	0,172
6	MCSSHA-7	256	1	8000000	0,172
7	MCSSHA-3	256	1000	10240	0,234
8	MCSSHA-4 (delay=2)	256	1000	10240	0,187
9	MCSSHA-4 (delay=3)	256	1000	10240	0,234
10	MCSSHA-5	256	1000	10240	0,234
11	MCSSHA-6	256	1000	10240	0,25
12	MCSSHA-7	256	1000	10240	0,25
13	MCSSHA-3	256	1000000	8	1,06
14	MCSSHA-4 (delay=2)	256	1000000	8	1,014
15	MCSSHA-4 (delay=3)	256	1000000	8	1,045
16	MCSSHA-5	256	1000000	8	1,732
17	MCSSHA-6	256	1000000	8	1,061
18	MCSSHA-7	256	1000000	8	2,496
19	MCSSHA-3	512	1	8000000	0,156

20	MCSSHA-4 (delay=2)	512	1	8000000	0,125
21	MCSSHA-4 (delay=3)	512	1	8000000	0,156
22	MCSSHA-5	512	1	8000000	0,156
23	MCSSHA-6	512	1	8000000	0,156
24	MCSSHA-7	512	1	8000000	0,171
25	MCSSHA-3	512	1000	10240	0,234
26	MCSSHA-4 (delay=2)	512	1000	10240	0,187
27	MCSSHA-4 (delay=3)	512	1000	10240	0,234
28	MCSSHA-5	512	1000	10240	0,234
29	MCSSHA-6	512	1000	10240	0,218
30	MCSSHA-7	512	1000	10240	0,234
31	MCSSHA-3	512	1000000	8	1,825
32	MCSSHA-4 (delay=2)	512	1000000	8	1,84
33	MCSSHA-4 (delay=3)	512	1000000	8	1,888
34	MCSSHA-5	512	1000000	8	3,183
35	MCSSHA-6	512	1000000	8	1,904
36	MCSSHA-7	512	1000000	8	4,29

4.2 Оценка сложности реализации алгоритма MCSSHA-7 в элементарных операциях

Под элементарными операциями будем понимать следующие:

1) Сложение и вычитание по модулю 256;
2) Замена байта по подстановке π;
3) Присваивание байту памяти некоторого значения.

Далее, для краткости, одну элементарную операцию также будем обозначать эл. оп.

4.2.1 Оценка объема памяти, необходимого для реализации MCSSHA-7.

Подсчитаем объем памяти, необходимой для реализации MCSSHA-7.

1) Память, необходимая для хранения подстановки π. Очевидно, что ее объем равен 256 байт.
2) Память для хранения контекста хеширования. Контекст хеширования может быть следующим:

```
#define MCSSHA7_LBLOCK        128
```

```
typedef struct MCSSHA7state_st
  {
  DataLength hashbitlen; // длина хеш-функции в битах
  DataLength SRbyteLen;  // длина регистра сдвига SR
  BitSequence x[6];      // вспомогательная память (SR points, remain bits)
  BitSequence data[MCSSHA7_LBLOCK];     // SR
  } MCSSHA7_CTX,hashState;
```

где, в соответствии с принятыми в NIST стандартами,

```
typedef unsigned char BitSequence;
typedef unsigned long long DataLength;
```

Таким образом, объем контекста составляет 152 байт, а общий объем требуемой памяти составляет 408 байт, с учетом вспомогательных массивов, использующихся в процессе вычисления MCSSHA-7, общий объем памяти не превосходит 1 Кб.

4.2.2 Число элементарных операций, необходимых для реализации этапа Preprocessing.

Этап Preprocessing сводится к начальному заполнению контекста хеширования. Поэтому количество элементарных операций на этом этапе равно размеру контекста, т.е. 152.

4.2.3 Число элементарных операций, необходимых для реализации одного такта работы SR.

Это число равно 4(число точек съема) + 1(замена по подстановке) + 1(запись результата в вспомогательный байт памяти) + 4 (сдвиг точек съема) + 1(замена в состоянии SR одной точки на результат из вспомогательной памяти). Таким образом, один такт работы SR требует 11 (такт задержки) или 12 (такт подачи байта сообщения на вход) элементарных операций. Всюду далее будем обозначать это число N_0 (такт задержки) или N_1 (такт подачи байта на вход).

4.2.4 Число элементарных операций, необходимых для реализации этапа pre-hash computation.

Этап pre-hash computation для сообщения длины M состоит из ¾ M тактов N_0 и ¼ M тактов N_1. Таким образом, общая трудоемкость этапа pre-hash computation не превосходит 12*M элементарных операций.

4.2.5 Число элементарных операций, необходимых для реализации этапа final-hash computation.

Трудоемкость этого этапа при длине SR на этапе pre-hash computation, равной N, и размере дайджеста, равной H, состоит из:

1) Переписывание во вспомогательный массив состояния SR – N эл. оп.
2) Заполнение состояния SR начальным состоянием – N эл. оп.
3) N тактов работы SR, т.е. 12*N эл. оп.
4) Реверсирование входной последовательности длины N. Трудоемкость – 5*N эл. оп.
5) N тактов работы SR, т.е. 12*N эл. оп.
6) Переписывание во вспомогательный массив состояния SR – N эл. оп.
7) Заполнение состояния SR начальным состоянием – H эл. оп.
8) N тактов работы SR, т.е. 12*N эл. оп.
9) Реверсирование входной последовательности длины N. Трудоемкость – 5*N эл. оп.
10) N тактов работы SR, т.е. 12*N эл. оп.

Таким образом, трудоемкость этапа final hash computation составляет 61*N + H эл. оп.

5. Контрольный пример функционирования MCSSHA-7 при H=32

Предположим, что для H = 32 необходимо вычислить дайджест сообщения M = “12345”. Красным цветом выделены SR points.

1) Pre-hash computation. N = 64.

№ такта	Знак на входе	SR state в начале такта
0	0x31	0001020304050607080 90A0B0C0D0E0F101112131415161718191A1B1C1D1E1F 202122232425262728292A2B2C2D2E2F303132333435363738393A3B3C3D3E3F
1	0	980102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F 202122232425262728292A2B2C2D2E2F303132333435363738393A3B3C3D3E3F
2	0	98270203040506070 8090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F 202122232425262728292A2B2C2D2E2F303132333435363738393A3B3C3D3E3F
3	0	9827DE0304050607080 90A0B0C0D0E0F101112131415161718191A1B1C1D1E1F 202122232425262728292A2B2C2D2E2F303132333435363738393A3B3C3D3E3F
4	0x32	9827DE790405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F 202122232425262728292A2B2C2D2E2F303132333435363738393A3B3C3D3E3F
5	0	9827DE79B3050607080 90A0B0C0D0E0F101112131415161718191A1B1C1D1E1F 202122232425262728292A2B2C2D2E2F303132333435363738393A3B3C3D3E3F
6	0	9827DE79B3AC060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F 202122232425262728292A2B2C2D2E2F303132333435363738393A3B3C3D3E3F
7	0	9827DE79B3ACC80708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F 202122232425262728292A2B2C2D2E2F303132333435363738393A3B3C3D3E3F
8	0x33	9827DE79B3ACC8460809 0A0B0C0D0E0F101112131415161718191A1B1C1D1E1F 202122232425262728292A2B2C2D2E2F303132333435363738393A3B3C3D3E3F
9	0	9827DE79B3ACC8465F090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F 202122232425262728292A2B2C2D2E2F303132333435363738393A3B3C3D3E3F
10	0	9827DE79B3ACC8465FF80A0B0C0D0E0F101112131415161718191A1B1C1D1E1F 202122232425262728292A2B2C2D2E2F303132333435363738393A3B3C3D3E3F

11	0	9827DE79B3ACC8465FF8360B0C0D0E0F101112131415161718191A1B1C1D1E1F 202122232425262728292A2B2C2D2E2F303132333435363738393A3B3C3D3E3F
12	0x34	9827DE79B3ACC8465FF8368B0C0D0E0F101112131415161718191A1B1C1D1E1F 202122232425262728292A2B2C2D2E2F303132333435363738393A3B3C3D3E3F
13	0	9827DE79B3ACC8465FF8368B100D0E0F101112131415161718191A1B1C1D1E1F 202122232425262728292A2B2C2D2E2F303132333435363738393A3B3C3D3E3F
14	0	9827DE79B3ACC8465FF8368B10FD0E0F101112131415161718191A1B1C1D1E1F 202122232425262728292A2B2C2D2E2F303132333435363738393A3B3C3D3E3F
15	0	9827DE79B3ACC8465FF8368B10FDCE0F10111213141516171819 1A1B1C1D1E1F 202122232425262728292A2B2C2D2E2F303132333435363738393A3B3C3D3E3F
16	0x35	9827DE79B3ACC8465FF8368B10FDCEAE1011121314151617181 91A1B1C1D1E1F 202122232425262728292A2B2C2D2E2F303132333435363738393A3B3C3D3E3F
17	0	9827DE79B3ACC8465FF8368B10FDCEAE8D1112131415161718191A1B1C1D1E1F 202122232425262728292A2B2C2D2E2F303132333435363738393A3B3C3D3E3F
18	0	9827DE79B3ACC8465FF8368B10FDCEAE8DC3121314151617181 91A1B1C1D1E1F 202122232425262728292A2B2C2D2E2F303132333435363738393A3B3C3D3E3F
19	0	9827DE79B3ACC8465FF8368B10FDCEAE8DC31C131415161718191A1B1C1D1E1F 202122232425262728292A2B2C2D2E2F303132333435363738393A3B3C3D3E3F
20		9827DE79B3ACC8465FF8368B10FDCEAE8DC31CB41415161718191A1B1C1D1E1F 202122232425262728292A2B2C2D2E2F303132333435363738393A3B3C3D3E3F

Этап Pre-hash computing завершен. Входная последовательность для Final hash computation - 9827DE79B3ACC8465FF8368B10FDCEAE8DC31CB41415161718191A1B1C1D1E1F 202122232425262728292A2B2C2D2E2F303132333435363738393A3B3C3D3E3F, ее длина – 64 байта.

2) Построение предварительного значения хеш-функции. N = 64. Длина входной последовательности – 128 байт.

№ такта	Знак на входе	SR state в начале такта
0	0x98	0001020304050607080 90A0B0C0D0E0F101112131415161718191A1B1C1D1E1F 202122232425262728292A2B2C2D2E2F303132333435363738393A3B3C3D3E3F
1	0x27	FF0102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F 202122232425262728292A2B2C2D2E2F303132333435363738393A3B3C3D3E3F
2	0xDE	FFD10203040506070 8090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F 202122232425262728292A2B2C2D2E2F303132333435363738393A3B3C3D3E3F
…	…	…
62	0x3E	FFD10AD247CD5D68007533E4AF49108A53D0982EDA2235AA3CB4A8CD1EDC19C8 E94958E663C0EADC5A8640E6D87DCBA47E618E1937E656D4ACE15C8CBB873E3F
63	0x3F	FFD10AD247CD5D68007533E4AF49108A53D0982EDA2235AA3CB4A8CD1EDC19C8 E94958E663C0EADC5A8640E6D87DCBA47E618E1937E656D4ACE15C8CBB877F3F
64	0x3F	FFD10AD247CD5D68007533E4AF49108A53D0982EDA2235AA3CB4A8CD1EDC19C8 E94958E663C0EADC5A8640E6D87DCBA47E618E1937E656D4ACE15C8CBB877F3A
65	0x3E	AAD10AD247CD5D68007533E4AF49108A53D0982EDA2235AA3CB4A8CD1EDC19C8 E94958E663C0EADC5A8640E6D87DCBA47E618E1937E656D4ACE15C8CBB877F3A
…	…	…
126	0x27	AA1F5A7726D000DF1E388649A13CCED38D4D136699763AD6CFA1D336EC209A2F BEE1C15E2A7D72333034A6C43FE882022FBCFC25762F5692C8F2522102D47F3A
127	0x98	AA1F5A7726D000DF1E388649A13CCED38D4D136699763AD6CFA1D336EC209A2F BEE1C15E2A7D72333034A6C43FE882022FBCFC25762F5692C8F252102D43A3A
128		AA1F5A7726D000DF1E388649A13CCED38D4D136699763AD6CFA1D336EC209A2F

		BEE1C15E2A7D72333034A6C43FE882022FBCFC25762F5692C8F2522102D43A61

Этап построения предварительного значения хеш-функции завершен. Входная последовательность для этапа окончательного вычисления хеш-функции –

AA1F5A7726D000DF1E388649A13CCED38D4D136699763AD6CFA1D336EC209A2F
BEE1C15E2A7D72333034A6C43FE882022FBCFC25762F5692C8F2522102D43A61,
ее длина – 64 байта.

3) Построение окончательного значения хеш-функции. N = 32. Длина входной последовательности – 128 байт.

№ такта	Знак на входе	SR state в начале такта
0	0xAA	0001020304050607080 90A0B0C0D0E0F101112131415161718191A1B1C1D1E1F
1	0x1F	110102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F
2	0x5A	118C020304050607080 90A0B0C0D0E0F101112131415161718191A1B1C1D1E1F
…	…	…
62	0x3A	181ADE013AC32D2F43262A54E2036F0B174F92DC09DF6CADB418B0D5C8FBC8EC
63	0x61	181ADE013AC32D2F43262A54E2036F0B174F92DC09DF6CADB418B0D5C8FBABEC
64	0x61	181ADE013AC32D2F43262A54E2036F0B174F92DC09DF6CADB418B0D5C8FBABFF
65	0x3A	E41ADE013AC32D2F43262A54E2036F0B174F92DC09DF6CADB418B0D5C8FBABFF
…	…	…
126	0x1F	448BA5FD01B7781F7C1593B6A28775C109C1B63C102D0F02CDF5F4A85421E98D
127	0xAA	448BA5FD01B7781F7C1593B6A28775C109C1B63C102D0F02CDF5F4A854213E8D
128		448BA5FD01B7781F7C1593B6A28775C109C1B63C102D0F02CDF5F4A854213E29

Таким образом, хеш-функцией от сообщения «12345» является значение 448BA5FD01B7781F7C1593B6A28775C109C1B63C102D0F02CDF5F4A854213E29 длиной 32 байта.

6. Контрольные тесты NIST

Одним из требований NIST к участникам конкурса SHA-3 было включение в заявку результатов специальных тестов, названных в [6] «A series of Known Answer Tests (KATs) and Monte Carlo Tests (MCTs)»

Ниже приводятся некоторые результаты этих тестов для MCSSHA-7.

##

ShortMsgKAT_256.txt

Algorithm Name: MCSSHA-7

Principal Submitter: Mikhail Maslennikov

##

Len = 0

Msg = 00

MD = 4DAF45A6D9661A2D435C33CCDEA306BA2A55740E99E78D9449B8750FE5B7E850

Len = 1

Msg = 00

MD = C0B3C621D3A8F2F577E925F98CE3768C34754294131DCC9E1EB3373833F1E3EB

Len = 2

Msg = C0

MD = 9FBF675FDBF90755DB8517B4E87783199557EA20FC1738BB975A77FF51AD2337

Len = 3

Msg = C0

MD = 8E512E3384B87C6D875C277C8ED5421430B3F19D1EA26FFE6C12DEBD10592E45

Len = 4

Msg = 80

MD = 71E718762444AB2CEDEE87FD21EBF7B380D56A0D3F0101349E3A2E6766D1ECA5

Len = 5

Msg = 48

MD = E7AECF6420E90AEE7560ACF23CA97B4FD033E065FF06BD6B064DEAB8C63A8809

Len = 6

Msg = 50

MD = 00F51560674FABAF6959C549571E11B653650B9A183291EDBCEBB06D4D54D385

Len = 7

Msg = 98

MD = E666A8A4EE0B94428DFD8702D20470554CD0C68CD6329EF783B6954A2A33A0C4

Len = 8

Msg = CC

MD = 56D01081AE52FF81E08619CD767E9B5F0D360CFD4A6F0C7BC714F9BA8EA025ED

##

ShortMsgKAT_512.txt

Algorithm Name: MCSSHA-7

Principal Submitter: Mikhail Maslennikov

##

Len = 0

Msg = 00

MD = C16CE21E660A45D1FE3DF5D922322CAC249A5C2D3DABD14E47B4C4CE17886FA554DAA967F390B4B88A88F7E1E40012A4C915979B7C2843DD4CCBC0E1E01A48AD

Len = 1

Msg = 00

MD = 9147BE204D0BDC47BEBAD732B17BA23290F9E98026D8BAE96AB551D937DEA2F05BF2B1712EC198AB25E6FC647A335748C09E856DC5F7B11CA017457834E45E56

Len = 2

Msg = C0

MD =
861F6EFA2D3DFD580EA3CABE3D3BB6147A3F9F7882F08320F2843526AE99786EB2DFE886182C9B41D1
04FC4059D79DE20C5A197405CE33F342245DE900FD289F

Len = 3

Msg = C0

MD =
24AD1D09F8154B795061603832D613B1522D7DD1422A0170B08D8E8F125A7B026ABA11586065D3EA5
364BCFFAAFB0F4A099D9F955A197B044661A271C28BD44B

Len = 4

Msg = 80

MD =
0A7194402771C3CBEB9D1600E0421521B30A130752691AE0BDFE69284AD0CB12E5C7AA8C597185971E
BA8B3C57B09346EFD7D1B6421944CAFE8920E31D1404CE

Len = 5

Msg = 48

MD =
7C43D1A08FE58598C6F314749034410DCF40AEE546E3ED420CE2BD2FD941494D0CC35DFB1CF4ADB9A
27CD0C1BA4B4E3018C9917046525D64A0DD5BAFD8A89629

Len = 6

Msg = 50

MD =
2E01CE934443E08FECFD7843CD2DFC7A8054BC45E42A8C762A4A04955ECFD9750037CBCD9C4D5BD59
21579FC4A7DA838C12B87247FA5CDADC264C4150A9E27E5

Len = 7

Msg = 98

MD =
DBEA3DD5465C98DC1E5F6F6EC690D3E3761AA442E1E1372B7E56949D4605C23833F2D1D95E0487FA4F
F70C51D72F8E4987A7AD04001AF2C075936C92F40384E1

Len = 8

Msg = CC

MD =
17ED52E413D71471B7F89AFB95A643A4113DF076E7F5CFBCD9CE84705AB9CB3A6193AC8385257B9BCF
5CC93981B5CA36BF945FB76F8551487943A9A4F4ED1861

##

LongMsgKAT_256.txt

Algorithm Name: MCSSHA-7

Principal Submitter: Mikhail Maslennikov

##

Len = 2048

Msg =
724627916C50338643E6996F07877EAFD96BDF01DA7E991D4155B9BE1295EA7D21C9391F4C4A41C75F
77E5D27389253393725F1427F57914B273AB862B9E31DABCE506E558720520D33352D119F699E784F9E
548FF91BC35CA147042128709820D69A8287EA3257857615EB0321270E94B84F446942765CE882B191F
AEE7E1C87E0F0BD4E0CD8A927703524B559B769CA4ECE1F6DBF313FDCF67C572EC4185C1A88E86EC11
B6454B371980020F19633B6B95BD280E4FBCB0161E1A82470320CEC6ECFA25AC73D09F1536F286D3F9
DACAFB2CD1D0CE72D64D197F5C7520B3CCB2FD74EB72664BA93853EF41EABF52F015DD591500D018D
D162815CC993595B195

MD = 19D664F7D84B3B1C648E0680E0D58770192B07F454CE1423C87CA5ECCE77701F

Len = 2111

Msg =
919FE5E7F35F64A7487649E564771DBBF10AE204ECC2181312D1A79FB579297C94F0DB9EAAE9E009A4
F02057AF2C973C5DAFA7B60154371A5D2C8E992FB6429176F8424B1A866BC1D1BED00438E97FAB4204
0DCACDEF7CA9FC2033059B8898BB40CCFB2634B051797BDF3B915C503EC81839AD01E0F4F2F871EFF2
008D40011730BE7A47888E7955A806876BE120CB0F3A139A3620154ECC6482A70F5629F6A9D3341BE6
FBBF48E5AA0C53589A04F057DD44268AFFCABF75ADFC549F73F454264D46A98CCA80E3000C7446853D
D5B430C9344E87E3230555B09FB3E7E64B5AD3989293AC0FEEC0E75F909696F028A5525D26DDEA5D2B
2C813FB3613DFF38CE23209285CC77C60860

MD = 23C50F435F71205E9ABE4E9BF7ECD9B17713BE1788D9BA461590E23D054FE61F

Len = 2174

Msg =
BD687B6D6684949FBD52A8196DF5D5927B403DEE88A47E4E4B52071BDDA6A6284A9EA1EB84950BA17
A60277B0FEA83C930E5619D5A6D232901B9D3A696662C7FD5F1349B494BC166C67B717C06D03E9FB05
DB9EE292D4792F459E0ACC08369E3D14684CFF388C940F30738FB71D97EFEB51C7DB553CB85B6FC184
29815EC71420822151BB8F5135DD698EFE3D3118963EE194F94682DFA62EC9D3BCB5F826DA79077970
4DF2AC5C8A9FA2B50A42A87ED2FBADF1E1C8E55689531786A9858DE9E88009E56873FD2D431FCFC99B
85574342DC7E78406678BDE94B9A06974DFF154EAC3BC7977AC7C123EBCDAC0641A69B792058BA373
EDD1BA45E3339C2ABC032993B25F4E3BFA3D5FF620C621D94C40F8

MD = E761A4347DB65ADAC25C649C1AF50F0EF8F2838989236735229ECC17CAEFCA31

Len = 2237

Msg =
D13409007A6B3242CBAD4CE01D9AED77EFEF0A7725B96A30097EA092FA1E49A136DE03F3F348464C9
BEA033216F380440C7EC81E284738DCC640D485ED32F5D7DD1CF936C1677CBB3CDD8E0E001783FD5A
3F388AE6EF7085FF9A22FA722C7FCCF4CDC239B200D5D11884B565D01C84E4D563C6623C241C5AE081
2CE8F9201BBADE48198378A3E232EF3E10C03022620A266D0A346A6A22F202333A94D2CEF495DBA102
B133B8237449C3350BF80EE51B6EE8A972CCFC3ED5FD7790444DA253922EEF6C611180F91DF1B6E58E8
43D318D94A958A017590B14D383C0F385F9CF2514E8AE1EA749795E10F991E3FB744C6030EF6B02989E
8EFFBF8E8BF31FC39B692C517C7C012ECBD0E0785BDEF4801C4E036C98C6CD0C9328

MD = 543A32006833EA75DEDE7783EBD1CB7E3DC9873B2757DB896DEA31BB877615D0

Len = 2300

Msg =
68F891C11459B2E9B71774E2B8A2A5C3C9CC36072E3498498496F1C7901684F3E9DACF13A3F1BAE221
40DEE5253732E4D4196B534F675264B53D38659727797F0A18910CCB5B48FE2346C2E998B6537357AF8
D15826FCB57BA804FE143E765F4680A0B28A9E3716A59ABC60EF253E357A4784FF1BE4680C82D79781
3CE50355D8FDB3C75936CC33E1717B99D9A8ADE9D0EA9172662B708EBCBC31C05064FE67B287C56D0
1C12411E9B890AD16238B36E192B15CD86D26F4EFDE5B523D71656F5CEA6CA73BDC04FEB973D303BB
AF4C0264761092E23220CEB8359FC91C1D918FB3F32DBBA92DEAC1C71DA8BC4BE806803F6E7970FA64
721C645EED4C2BED7EEFA2B720931FDBD6C67B83756B1CE51C51F839C250AD900B9D49FE5188FC4A2B
5D0

MD = C0A1E3C398F4F2207E2E5601ECB08DB212192D893A2757A42D3265C55E8D1CE6

Len = 2363

Msg =
7D9222C58DC49F14BFD7198A9A1C338D17201C007822A91DCD262860364CA1DA8C0056033EB58E406
E36D5A4F6F7E9D98BE57126C9FE7676B58FDF0F9899432FF78DE2AB65C9000071EBA6967123F97BA3D

C1F3825D3C8C5148EAD7AEF0334F40CB0F6B982CB7C99CA39E1B4B2E3A3541683D1EC5560466F52175
518390D38C7461C116DB2D56EF913784D2E8B20959BBD6F8F3282C597D94A1EDBFA8F25089E9D2E8D
C465EAD90FA23E4388BD6BAFE22632749B7AAEA53D5CBCF9678227ADC3F4109F1849AC2539B6F2B25
B4D8EDAB41E8BFCFC337728DC48A8EA5119115FC1B133300D68231C96A9D518F6DC3CA51581309C53
F49510FE18F608A215069D41F2CFC84E53A9347FD723DFF9D3F5006F7C0B18441A29BDEF7725920260B
1613C6532A0A994B488E0

MD = 1873224FE49CE85141F9562F51FCED5112FBEF7E8AA99250C5E65EDA60979354

Len = 2426

Msg =
FC447A550FC565F964337521DBD6481FF6C4FEC25A2BC946230C0021570E75B0E3D50320AB24C4949C
B4EA7ED6D14D10E1EB9CB4116461A743D49F337597F12D09945285647B249A2AE3EBF69F1FC62A3653
2B2FC89ECED5B48AC27A0E18AF8B0F023BE5C00FF0BF8C16F412B34D4AE9ECD2963583FA268ED439B4
640FFCC57384EF066362E23D2F0712F9658CB7B7F56F548C4A3C7DB51D8FF4430D14CE1042221254CA
A4BF009B197E6F74B42067E95E42D6C8F2D37AA5522F5594D61745BA28C8F302E007316282985730FB
CB8BDAB0C5A3693BEEC32C0D57AD20D3394B7AC72831FF5DDFC8208E09057AB9E9EE2FAC208E9FCB6
D0B567C4DDED41ED286678DB3860B230C9A52C577867EC35A17EF902E258CD8314F36875D97543088
679415FD0D7C2A9CB6CAEE76A2A1DC294700

MD = 17C0EB07A1813EDC29E0392274EE83967FEBCDD4C2FCF0261682041D67C8D3AD

Len = 2489

Msg =
3F989C5FAAB635E462C4181FE623B8205FEBCA482B0613B8CB702D0C47CABC705DA23CD238FEAE5A7
9B3D630C22F12FCB845FC633882BC7A46058F945E6376A5DF954953A82D5C6723156BE49438431529C
9ED7A5ABC0E251EC9746F34196BFCB64850CDD962B4D4C0C37DFD7DEB4DADF6FA98F727DBE430A232
BE68161A834100BF81412B9BC4B34C7E1FC4CA216A7A04E23B5A1E0AFF6967465E8FEB63A7B450026F
B944C62ADA108A15FFF01FFB872925EE97EA314D84245E31745C7A3D3C83A59BD6F38205812B5CBB14
1833D381CA4DD3B261DAF19323BB790D0DE8F7EF8A28D1F2131B92DFA724B76BF6162872CFBAB0E3D
426741F8946CE449787896308A0DA3E5B5F087E96470A6990681F10F37DBDD10A799032CEAA9CDF8DC
0DE302E98980718C3F1C7D950A9D37216A3686211A658312B3300

MD = ED52352E5BE929E9B435C7531E7024AE35C551FB53BCF287945F3695D5B59DFF

##

LongMsgKAT_512.txt

Algorithm Name: MCSSHA-7

Principal Submitter: Mikhail Maslennikov

##

Len = 2048

Msg =
724627916C50338643E6996F07877EAFD96BDF01DA7E991D4155B9BE1295EA7D21C9391F4C4A41C75F
77E5D27389253393725F1427F57914B273AB862B9E31DABCE506E558720520D33352D119F699E784F9E
548FF91BC35CA147042128709820D69A8287EA3257857615EB0321270E94B84F446942765CE882B191F
AEE7E1C87E0F0BD4E0CD8A927703524B559B769CA4ECE1F6DBF313FDCF67C572EC4185C1A88E86EC11
B6454B371980020F19633B6B95BD280E4FBCB0161E1A82470320CEC6ECFA25AC73D09F1536F286D3F9
DACAFB2CD1D0CE72D64D197F5C7520B3CCB2FD74EB72664BA93853EF41EABF52F015DD591500D018D
D162815CC993595B195

MD =
F83F39803492FF5E36E65704ED6FB5DC08C624F3F01EF35C4D7B2732E289CE5BE03C50DC05D5DE0F7E5
A5C228678ACB20B503FE7D23282354B39D454D71C059D

Len = 2111

Msg =
919FE5E7F35F64A7487649E564771DBBF10AE204ECC2181312D1A79FB579297C94F0DB9EAAE9E009A4
F02057AF2C973C5DAFA7B60154371A5D2C8E992FB6429176F8424B1A866BC1D1BED00438E97FAB4204
0DCACDEF7CA9FC2033059B8898BB40CCFB2634B051797BDF3B915C503EC81839AD01E0F4F2F871EFF2
008D40011730BE7A47888E7955A806876BE120CB0F3A139A3620154ECC6482A70F5629F6A9D3341BE6
FBBF48E5AA0C53589A04F057DD44268AFFCABF75ADFC549F73F454264D46A98CCA80E3000C7446853D
D5B430C9344E87E3230555B09FB3E7E64B5AD3989293AC0FEEC0E75F909696F028A5525D26DDEA5D2B
2C813FB3613DFF38CE23209285CC77C60860

MD =
B59C5988902A5990114DBA20006BA0582362F4C7B73FD81C623FE70829A0D47DFF3F3B59F7681707C3
929A4247F1DEBE8AD6F624FBE0AB81F892FFEFF1753E28

Len = 2174

Msg =
BD687B6D6684949FBD52A8196DF5D5927B403DEE88A47E4E4B52071BDDA6A6284A9EA1EB84950BA17
A60277B0FEA83C930E5619D5A6D232901B9D3A696662C7FD5F1349B494BC166C67B717C06D03E9FB05
DB9EE292D4792F459E0ACC08369E3D14684CFF388C940F30738FB71D97EFEB51C7DB553CB85B6FC184
29815EC71420822151BB8F5135DD698EFE3D3118963EE194F94682DFA62EC9D3BCB5F826DA79077970
4DF2AC5C8A9FA2B50A42A87ED2FBADF1E1C8E55689531786A9858DE9E88009E56873FD2D431FCFC99B
85574342DC7E78406678BDE94B9A06974DFF154EAC3BC7977AC7C123EBCDAC0641A69B792058BA373
EDD1BA45E3339C2ABC032993B25F4E3BFA3D5FF620C621D94C40F8

MD =
8DC31879507AA12BB135BBA0668D35659F2A1B715A8FF69CAFEA80FFBEADE2FBA783736C8B1A0041F
4D51BEC538ACD3FF806535FB086E7427AB87ECAEAF4FE07

Len = 2237

Msg =
D13409007A6B3242CBAD4CE01D9AED77EFEF0A7725B96A30097EA092FA1E49A136DE03F3F348464C9
BEA033216F380440C7EC81E284738DCC640D485ED32F5D7DD1CF936C1677CBB3CDD8E0E001783FD5A
3F388AE6EF7085FF9A22FA722C7FCCF4CDC239B200D5D11884B565D01C84E4D563C6623C241C5AE081
2CE8F9201BBADE48198378A3E232EF3E10C03022620A266D0A346A6A22F202333A94D2CEF495DBA102
B133B8237449C3350BF80EE51B6EE8A972CCFC3ED5FD7790444DA253922EEF6C611180F91DF1B6E58E8
43D318D94A958A017590B14D383C0F385F9CF2514E8AE1EA749795E10F991E3FB744C6030EF6B02989E
8EFFBF8E8BF31FC39B692C517C7C012ECBD0E0785BDEF4801C4E036C98C6CD0C9328

MD =
5A50B14BB496059D88434308F179B32B0A3B3F57E52F4D3E274548558913B7BBB57592619826821B36
551CC3E5915BFEDF94D55791803AB32A6BC04D35530947

Len = 2300

Msg =
68F891C11459B2E9B71774E2B8A2A5C3C9CC36072E3498498496F1C7901684F3E9DACF13A3F1BAE221
40DEE5253732E4D4196B534F675264B53D38659727797F0A18910CCB5B48FE2346C2E998B6537357AF8
D15826FCB57BA804FE143E765F4680A0B28A9E3716A59ABC60EF253E357A4784FF1BE4680C82D79781
3CE50355D8FDB3C75936CC33E1717B99D9A8ADE9D0EA9172662B708EBCBC31C05064FE67B287C56D0
1C12411E9B890AD16238B36E192B15CD86D26F4EFDE5B523D71656F5CEA6CA73BDC04FEB973D303BB
AF4C0264761092E23220CEB8359FC91C1D918FB3F32DBBA92DEAC1C71DA8BC4BE806803F6E7970FA64
721C645EED4C2BED7EEFA2B720931FDBD6C67B83756B1CE51C51F839C250AD900B9D49FE5188FC4A2B
5D0

MD =
AC6F00EBCA0A8268EBD61C01DF117CD0DD6A03F6DE628ED27A6F2D6C063D066EF82CD21B2703EB9E2
590F52756E6C6832C1A8F19D0930D189FB70A59E9E4DE5B

Len = 2363

Msg =
7D9222C58DC49F14BFD7198A9A1C338D17201C007822A91DCD262860364CA1DA8C0056033EB58E406
E36D5A4F6F7E9D98BE57126C9FE7676B58FDF0F9899432FF78DE2AB65C9000071EBA6967123F97BA3D
C1F3825D3C8C5148EAD7AEF0334F40CB0F6B982CB7C99CA39E1B4B2E3A3541683D1EC5560466F52175
518390D38C7461C116DB2D56EF913784D2E8B20959BBD6F8F3282C597D94A1EDBFA8F25089E9D2E8D
C465EAD90FA23E4388BD6BAFE22632749B7AAEA53D5CBCF9678227ADC3F4109F1849AC2539B6F2B25
B4D8EDAB41E8BFCFC337728DC48A8EA5119115FC1B133300D68231C96A9D518F6DC3CA51581309C53

F49510FE18F608A215069D41F2CFC84E53A9347FD723DFF9D3F5006F7C0B18441A29BDEF7725920260B
1613C6532A0A994B488E0

MD =
FE5CDC246D955F17DBA870EC3C98F9F36105DA45BD68A622B5DE51C864506DEFA97C11BF4D164D86A
61AE1EAE820301C77CD93FA8D7B8DD7E76686C683EDB52C

Len = 2426

Msg =
FC447A550FC565F964337521DBD6481FF6C4FEC25A2BC946230C0021570E75B0E3D50320AB24C4949C
B4EA7ED6D14D10E1EB9CB4116461A743D49F337597F12D09945285647B249A2AE3EBF69F1FC62A3653
2B2FC89ECED5B48AC27A0E18AF8B0F023BE5C00FF0BF8C16F412B34D4AE9ECD2963583FA268ED439B4
640FFCC57384EF066362E23D2F0712F9658CB7B7F56F548C4A3C7DB51D8FF4430D14CE1042221254CA
A4BF009B197E6F74B42067E95E42D6C8F2D37AA5522F5594D61745BA28C8F302E007316282985730FB
CB8BDAB0C5A3693BEEC32C0D57AD20D3394B7AC72831FF5DDFC8208E09057AB9E9EE2FAC208E9FCB6
D0B567C4DDED41ED286678DB3860B230C9A52C577867EC35A17EF902E258CD8314F36875D97543088
679415FD0D7C2A9CB6CAEE76A2A1DC294700

MD =
E9CE769D3C5D0915232184AD643EED80F78BC08C88599233F26C73EFAB4C12C969EB96721D6F90A125
9B3A0BDCD1AD6C272445B177181389EB54854DBBCE133B

Len = 2489

Msg =
3F989C5FAAB635E462C4181FE623B8205FEBCA482B0613B8CB702D0C47CABC705DA23CD238FEAE5A7
9B3D630C22F12FCB845FC633882BC7A46058F945E6376A5DF954953A82D5C6723156BE49438431529C
9ED7A5ABC0E251EC9746F34196BFCB64850CDD962B4D4C0C37DFD7DEB4DADF6FA98F727DBE430A232
BE68161A834100BF81412B9BC4B34C7E1FC4CA216A7A04E23B5A1E0AFF6967465E8FEB63A7B450026F
B944C62ADA108A15FFF01FFB872925EE97EA314D84245E31745C7A3D3C83A59BD6F38205812B5CBB14
1833D381CA4DD3B261DAF19323BB790D0DE8F7EF8A28D1F2131B92DFA724B76BF6162872CFBAB0E3D
426741F8946CE449787896308A0DA3E5B5F087E96470A6990681F10F37DBDD10A799032CEAA9CDF8DC
0DE302E98980718C3F1C7D950A9D37216A3686211A658312B3300

MD =
FF6D36F66F78D51C11BCFEB1C964F6DA9D58F5BBCA0141A8B5780F4E01BCBB9B718D3B69F56A9B5B00
A81ED802B916A61C31438F001EFB3361BCC754D1B36DCE

##

ExtremelyLongMsgKAT_256.txt

Algorithm Name: MCSSHA-7

Principal Submitter: Mikhail Maslennikov

##

Repeat = 16777216

Text = abcdefghbcdefghicdefghijdefghijkefghijklfghijklmghijklmnhijklmno

MD = 6F618D31CFE93B33E215D6051F16F6E9EA053329A054DFF619829B2074F2628D

##

ExtremelyLongMsgKAT_512.txt

Algorithm Name: MCSSHA-7

Principal Submitter: Mikhail Maslennikov

##

Repeat = 16777216

Text = abcdefghbcdefghicdefghijdefghijkefghijklfghijklmghijklmnhijklmno

MD =
4E9A7DA9880073BEFD2A7E1764703BC1D141904DCF441495F123F858F68AE9DF7FD4E02DB147D25CD
F473A452DCA91A81AC25299DE4524B881D395D201420398

##

MonteCarlo_256.txt

Algorithm Name: MCSSHA-7

Principal Submitter: Mikhail Maslennikov

##

Seed =
6CD4C0C5CB2CA2A0F1D1AECEBAC03B52E64EA03D1A1654372936545B92BBC5484A59DB74BB60F9C40
CEB1A5AA35A6FAFE80349E14C253A4E8B1D77612DDD81ACE926AE8B0AF6E53176DBFFCC2A6B88C6BD
765F939D3D178A9BDE9EF3AA131C61E31C1E42CDFAF4B4DCDE579A37E150EFBEF5555B4C1CB40439D
835A724E2FAE7

j = 0

MD = 9AC8B92958C94F77EE58AF2E1A62CC41F03311B48728696FF8193F9BA9E669A8

j = 1

MD = 68DD081A9AC0028AD186D6FD1981D30B2EE2B3A42EA9BA8A243934F9AF2C979A

j = 2

MD = 0C21E1A803A689D28B6B8355671731F9103EEB09035170A7FB290AA77D18D049

j = 3

MD = 392DE9A61B49ABC611DF5BA72B4441C25178841A29515CFE7F270D22F9046886

j = 4

MD = 98B9BE37A6DCA56E669DD9C4485183EFB15F34698C23A5C163B81ACD437C99DE

##

MonteCarlo_512.txt

Algorithm Name: MCSSHA-7

Principal Submitter: Mikhail Maslennikov

##

Seed =
6CD4C0C5CB2CA2A0F1D1AECEBAC03B52E64EA03D1A1654372936545B92BBC5484A59DB74BB60F9C40
CEB1A5AA35A6FAFE80349E14C253A4E8B1D77612DDD81ACE926AE8B0AF6E53176DBFFCC2A6B88C6BD
765F939D3D178A9BDE9EF3AA131C61E31C1E42CDFAF4B4DCDE579A37E150EFBEF5555B4C1CB40439D
835A724E2FAE7

j = 0

MD =
26A47750295E522C2878357A946F9B7C18185C942642CAE22F8B9F50CDE1B12412917007E1CEA95038
D53746FC665067B3E870AF9C36A7157C58070489D5AA85

j = 1

MD =
BE86000CE8A956C0E1A424462BBFD418BF68A7537B8E73DF8C6CB4D9531D345F4652013C3897FF62C5
2C36B6E310248B21088BDF81067A623880A3422E87CE1E

j = 2

MD =
BA35CFAA63FC8D24E58B3F4F97B2954D96F885254B77E4F74E41BDAC34293E3C2ED12D6CD871769922
FDE72119E390F44F0A3A327B2DE69312B5D8ADFE365737

j = 3

MD =
FB1D8926FE111D692607A4EA18CA8057B1BBBAED9F1C46BC94F24A312C9CD560CFFEC77330EECA8A0
28861C82903CF00A57C747861475C647EC41F8A156396AB

j = 4

MD =
616989E104364A2529A964599E31488E8D1784FD4FD8901128B010B7B9BF8AE994A79D1907F5DCAFDC
6CFBC0AC5DC5CC7757484E4A095FBA5637BEE5E57A9AD0

7. Области возможного применения MCSSHA-7

Учитывая все приведенные выше данные, алгоритм хеширования MCSSHA-7 может использоваться как в качестве функции хеширования, так и как способ быстрой выработки имитоприставки и средства защиты компьютерного файла от случайных искажений. Также алгоритм MCSSHA-7 может использоваться в банковской сфере для выработки кодов подтверждения достоверности (КПД) платежных документов, в которых используются ручные операции и длина КПД строго ограничена. Сам КПД зависит как от содержания платежного документа (информационный блок), так и от некоторого или некоторых симметричных ключей. Информационный блок и все ключи подаются на вход MCSSHA-7 при длине дайджеста H равной 4 или 5. КПД в этом случае будет представлять из себя 8 или 10 hex-символов. Такая ситуация в настоящее время имеет место в ЦБ РФ при обработке почтовых и телеграфных авизо. Работающий в настоящее время в ЦБ РФ алгоритм выработки КПД разрабатывался свыше 20 лет назад и был ориентирован на морально устаревший калькулятор «Электроника-МК 85 С». Переход к MCSSHA-7 позволит реализовать алгоритм выработки КПД, например, на смарт-карте и осуществлять все операции с ключами также внутри смарт-карты. Подобный подход позволит значительно повысить безопасность банковских переводов с помощью почтовых и телеграфных авизо.

Литература

1. Mikhail Maslennikov. Secure hash algorithm MCSSHA-3. Revision: September 23, 2008
2. Mikhail Maslennikov. Secure hash algorithm MCSSHA-4. Revision: January 08, 2009
3. Mikhail Maslennikov. Secure hash algorithm MCSSHA-5. Revision: June 04, 2009
4. Mikhail Maslennikov. Secure hash algorithm MCSSHA-6. Revision: June 08, 2009
5. Jean-Philippe Aumasson and Mar´ıa Naya-Plasencia. Cryptanalysis of the MCSSHA Hash Functions.
6. Announcing Request for Candidate Algorithm Nominations for a New Cryptographic Hash Algorithm (SHA–3) Family
7. Handbook of Applied Cryptography, by A. Menezes, P. van Oorschot, and S. Vanstone, CRC Press, 1996.
8. Federal Information Processing Standards Publication (FIPS) 180-2. August 2002.
9. Национальный стандарт Российской Федерации ГОСТ Р 34.11 – 20. Информационные технологии. Криптографическая защита информации. Функция хеширования. Издание официальное. Москва. Стандартинформ 20.