Перевод FIPS-180-3 (SHS)

Версия 2

Контроль версий

Версия	Дата	Изменения
1	март 2010	Первая версия
2	май 2013	Незначительные изменения в оформлении

Будьте осторожны, это любительский перевод ! Связь по почте TiuTa сбк mail тчк ru

FIPS PUB 180-3

ПУБЛИКАЦИЯ ФЕДЕРАЛЬНЫХ СТАНДАРТОВ ОБРАБОТКИ ИНФОРМАЦИИ

СТАНДАРТ БЕЗОПАСНОГО ХЭША (SHS)

КАТЕГОРИЯ: КОМПЬЮТЕРНАЯ БЕЗОПАСНОСТЬ ПОДКАТЕГОРИЯ: КРИПТОГРАФИЯ

Лаборатория информационных технологий
национального института стандартов и технологий
Gaithersburg, MD 20899-8900

Октябрь 2008

Министерство торговли США
Carlos M. Gutierrez, министр

Национальный институт стандартов и технологий
Patrick Gallagher, и. о. директора

Вводная часть

Федеральные стандарты обработки информации национального института стандартов и технологий (NIST) являются официальными документами, связанными со стандартами и руководящими документами, принимаемыми и распространяемыми согласно федеральному закону об управлении информационной безопасностью (FISMA) от 2002 года.

Комментарии федеральных стандартов обработки информации приветствуются и принимаются директором лаборатории информационных технологий национального института стандартов и технологий по адресу 100 Bureau Drive, Stop 8900, Gaithersburg, MD 20899-8900.

Cita Furlani, директор
лаборатории информационных технологий

Реферат

Данный стандарт описывает пять хэш-алгоритмов, которые могут быть использованы для генерации отпечатков сообщений. Отпечатки используются для того, чтобы определить было ли сообщение изменено с момента вычисления отпечатка.

Ключевые слова: компьютерная безопасность, криптография, отпечаток сообщения, хэш-функция, хэш-алгоритм, федеральные стандарты обработки информации, стандарт безопасного хэша.

Федеральные стандарты обработки информации
Публикация 180-3
Октябрь 2008

Общая характеристика
СТАНДАРТА БЕЗОПАСНОГО ХЭША

Публикации федеральных стандартов обработки информации (FIPS PUBS) выпускаются национальным институтом стандартов и технологий (NIST) после утверждения министром торговли. Утверждение выполняется в соответствии с частью 5131 закона о реформе управления информационными технологиями от 1996 года (общественное право 104-106) и законом о компьютерной безопасности от 1987 года (общественное право 100-235).

1. Название стандарта. Стандарт безопасного хэша (SHS или FIPS PUB 180-3)

2. Категория стандарта. Стандарт компьютерной безопасности, криптография.

3. Пояснение. Данный стандарт описывает пять безопасных хэш-алгоритмов – SHA-1, SHA-224, SHA-256, SHA-384 и SHA-512 – предназначенных для получения сжатого представления электронных данных (сообщений). Когда сообщение с длиной меньшей чем 2^{64} бит (для SHA-1, SHA-224 и SHA-256) или меньшей 2^{128} бит (для SHA-384 и SHA-512) подаётся на вход хэш-алгоритма, то получаемый результат называется отпечатком сообщения. В зависимости от алгоритма отпечаток сообщения имеет длину от 160 до 512 бит. Безопасные хэш-алгоритмы обычно используются совместно с другими криптографическими алгоритмами, такими как алгоритмы цифровой подписи, коды аутентификации сообщения, использующие хэш-функцию и ключ, или в генераторах случайных чисел (бит).

Пять хэш-алгоритмов, описанных в данном стандарте, называют безопасными, потому что для данных алгоритмов вычислительно невозможно:

1) найти сообщение, соответствующее заданному отпечатку
2) найти два разных сообщения, отпечатки которых совпадают

Любое изменение сообщения повлечёт (с очень большой вероятностью) изменение его отпечатка. Это повлечёт неудачу при проверке, когда безопасный хэш-алгоритм используется совместно с алгоритмом цифровой подписи или алгоритмом вычисления кодов аутентификации сообщения, использующих хэш-функцию и ключ.

Настоящий стандарт принят взамен FIPS 180-2 [FIPS 180-2].

4. Утверждающий орган. Министр торговли.

5. Поддерживающая организация. Министерство торговли США, национальный институт стандартов и технологий (NIST), лаборатория информационных технологий (ITL).

6. Применимость. Данный стандарт разрешается применять во всех федеральных департаментах и агенствах для защиты важной несекретной информации, на которую не распространяется действие статьи 2315 раздела 10 Кодекса США (10 USC 2315) и не находящейся в пределах национальной системы безопасности, как определено в статье 3502(2) раздела 44 Кодекса США (44 USC 3502(2)). Данный стандарт должен быть применён при необходимости использования безопасного хэш-алгоритма в федеральных задачах, в том числе совместно с другими криптографическими алгоритмами и протоколами. Разрешается внедрение и использование данного стандарта в частных и коммерческих организациях.
7. Подробные обозначения. Федеральный стандарт обработки информации (FIPS) 180-3 и дополнительное – стандарт безопасного хэша (SHS).
8. Реализации. Описанные здесь безопасные хэш-алгоритмы могут быть реализованы в виде программ, прошивок, аппаратных средств или в любых их комбинациях. Реализации алгоритмов будут считаться соответствующими данному стандарту, только если это подтверждено NIST. Информация о программе подтверждения доступна по адресу http://csrc.nist.gov/groups/STM/index.html.
9. График внедрения. Руководство по тестированию и проверке на соответствие FIPS 180-3, а также взаимосвязь стандартов FIPS 180-3 и FIPS 140-2, могут быть найдены в подразделе 1.10 руководства по реализации стандарта FIPS PUB 140-2 и реализации программы проверки криптографических модулей по адресу http://csrc.nist.gov/groups/STM/cmvp/index.html.
10. Патенты. Реализации безопасных хэш-алгоритмов данного стандарта могут быть защищены патентами США или иностранными патентами.
11. Экспортный контроль. Некоторые криптографические устройства и относящиеся к ним технические данные являются объектом федерального экспортного контроля. Экспорт криптографических модулей, реализующих данный стандарт, и относящихся к ним технических данных должен производиться согласно действующему федеральному законодательству и лицензии от бюро экспортного контроля министерства торговли США. Информация о правилах экспорта доступна по адресу http://www.bis.doc.gov/index.htm.
12. Уточнения. Несмотря на то, что целью данного стандарта является установление общих требований безопасности при генерации отпечатка сообщения, соответствие данному стандарту не гарантирует, что конкретная реализация будет безопасной. В каждом агентстве или департаменте ответственный орган должен гарантировать, что суммарно реализация обеспечивает приемлемый уровень безопасности. Данный стандарт будет пересматриваться каждые пять лет для того, чтобы оценить его адекватность.
13. Процедура отказа. В соответствии с распоряжением министра торговли выполнение федеральных стандартов обработки информации (FIPS) является обязательным. Федеральный закон об управлении информационной безопасностью (FISMA) не предусматривает возможности отказа от выполнения стандартов.
14. Где получить копию данного стандарта. Данная публикация доступна в электронном виде по адресу http://csrc.nist.gov/publications/. На этом же веб-сайте доступны другие публикации, посвящённые компьютерной безопасности.

Федеральные стандарты обработки информации
Публикация 180-3

Описание
СТАНДАРТА БЕЗОПАСНОГО ХЭША

Оглавление

1. Введение

Данный стандарт описывает пять алгоритмов вычисления безопасного хэша: SHA-1, SHA-224, SHA-256, SHA-384 и SHA-512. Все пять алгоритмов являются итеративными, однонаправленными хэш-функциями, которые могут обрабатывать сообщение для получения его сжатого представления, называемого отпечатком. Эти алгоритмы дают возможность определить, было ли сообщение изменено, так как любое изменение сообщения с большой вероятностью приведёт к изменению его отпечатка. Это свойство полезно при генерации и проверке цифровых подписей, кодов аутентификации сообщений, а также при генерации случайных чисел или бит.

Описание каждого алгоритма может быть разделено на две стадии: предварительная обработка и вычисление хэша.

Предварительная обработка включает в себя дополнение сообщения, разделение дополненного сообщения на блоки из m бит и присваивание начальных значений величинам, используемым при вычислении хэша.

На стадии вычисления хэша из дополненного сообщения формируется последовательность блоков данных. С помощью этой последовательности, функций, констант и пословных операций итеративно вычисляется ряд значений хэша. Последнее значение хэша, вычисленное на этой стадии, используется для получения отпечатка сообщения.

Наиболее значительно пять алгоритмов отличаются стойкостью, которую они обеспечивают для захэшированных данных. Стойкость этих пяти хэш-функций, а также стойкость систем в целом, образованных каждой из этих функций и другими криптографическими алгоритмами (такими как алгоритмы цифровой подписи; коды аутентификации сообщения, использующие хэш-функцию и ключ), обсуждается в [SP 800-57] и [SP 800-107].

Кроме того, пять алгоритмов различаются исчислением размера блоков и слов данных, которые используются в процессе хэширования. В таблице 1 представлены основные свойства этих хэш-алгоритмов.

Таблица 1.

Алгоритм	Размер сообщения, бит	Размер блока, бит	Размер слова, бит	Размер отпечатка, бит
SHA-1	$< 2^{64}$	512	32	160
SHA-224	$< 2^{64}$	512	32	224
SHA-256	$< 2^{64}$	512	32	256
SHA-384	$< 2^{128}$	1024	64	384
SHA-512	$< 2^{128}$	1024	64	512

2. Определения

2.1 Словарь терминов и сокращений

Бит	двоичная цифра, имеющая значение 0 или 1
Байт	совокупность 8 бит
FIPS	федеральный стандарт обработки информации
NIST	национальный институт стандартов и технологий
SHA	алгоритм вычисления безопасного хэша
SP	специальная публикация
Слово	совокупность 32 бит (4 байт) или 64 бит (8 байт), в зависимости от алгоритма безопасного хэша

2.2 Параметры алгоритма, знаки и термины

2.2.1 Параметры

В данном стандарте при описании алгоритма вычисления безопасного хэша используются следующие параметры:

a, b, c, …, h	рабочие переменные, являющиеся *w*-битными словами. Используются при вычислении значений хэша $H^{(i)}$
$H^{(i)}$	*i*-ое значение хэша. $H^{(0)}$ - это начальное значение хэша. $H^{(N)}$ - это конечное значение хэша, используемое для получения отпечатка сообщения.
$H_j^{(i)}$	*j*-ое слово *i*-ого значения хэша. $H_0^{(i)}$ - это крайнее левое слово *i*-ого хэш значения.
K_t	константа, используемая на *t*-ой итерации вычисления хэша
k	число нулей, дописываемых после сообщения на шаге его дополнения
ℓ	длина сообщения *M* в битах
m	число бит в блоке сообщения $M^{(i)}$
M	сообщение, подлежащее хэшированию
$M^{(i)}$	*i*-й блок сообщения размером *m* бит
$M_j^{(i)}$	*j*-ое слово *i*-ого блока сообщения. $M_0^{(i)}$ - это крайнее левое слово *i*-ого блока сообщения.
n	число бит, на которое происходит смещение или циклический сдвиг, при выполнении операции над словом
N	число блоков в дополненном сообщении
T	временное *w*-битное слово, используемое при вычислении хэша
w	число бит в слове
W_t	*t*-ое *w*-битное слово в последовательности блоков данных

2.2.2 Знаки и операции

При описании алгоритма вычисления безопасного хэша используются следующие знаки (каждый оперирует *w*-битными словами):

$\wedge$	операция побитового И
$\vee$	операция побитового ИЛИ («включающее ИЛИ»)
$\oplus$	операция побитового XOR («исключающее ИЛИ»)
$\neg$	операция побитового отрицания
+	сложение по модулю 2^w
<<	сдвиг влево. x<<n получается отбрасыванием *n* крайних левых бит слова *x* и последующим приписыванием к результату *n* нулевых бит справа
>>	сдвиг вправо. x>>n получается отбрасыванием *n* крайних правых бит слова *x* и последующим приписыванием к результату *n* нулевых бит слева

При описании алгоритма вычисления безопасного хэша используются следующие операции:

$ROTL^n(x)$	операция циклического сдвига влево, где *x* – это *w*-битное слово, *n* – целое в пределах $0 \le n < w$. Определяется как $ROTL^n(x) = (x << n) \vee (x >> w - n)$.

$ROTR^n(x)$ операция циклического сдвига вправо, где x – это w-битное слово, n – целое в пределах $0 \le n < w$. Определяется как $ROTR^n(x) = (x >> n) \vee (x << w - n)$.

$SHR^n(x)$ операция сдвига вправо, где x – это w-битное слово, n – целое в пределах $0 \le n < w$. Определяется как $SHR^n(x) = x >> n$.

3. Обозначения и соглашения

3.1 Битовые строки и целые

Для описания битовых строк и целых чисел будет использоваться следующая терминология:
1. *16-ричная цифра* – это элемент из множества {0, 1, …, 9, A, …, F}. 16-ричная цифра является представлением строки из 4 бит. Например, 16-ричная цифра «7» представляет 4-битную строку «0111», а 16-ричная цифра «A» представляет 4-битную строку «1010».
2. *Слово* – это строка из w бит, которая может быть представлена как последовательность 16-ричных цифр. Для преобразования слова в 16-ричные цифры, каждая 4-битная строка преобразуется в 16-ричную цифру, как описано в пункте 1 выше. Например, 32-битная строка

1010 0001 0000 0011 1111 1110 0010 0011

может быть представлена как «A103FE23». А 64-битная строка

1010 0001 0000 0011 1111 1110 0010 0011
0011 0010 1110 1111 0011 0000 0001 1010

может быть представлена как «A103FE2332EF301A».

На всём протяжении данного описания при представлении 32-битных и 64-битных слов будет использоваться соглашение о «тупоконечном» порядке расположения бит. То есть в пределах каждого слова наиболее значимый бит будет находиться в крайней левой позиции.

3. *Целое число* может быть представлено словом или парой слов. Для способа дополнения сообщения, описанного в подразделе 5.1, необходимо представление длины сообщения ℓ (в битах) в виде слова.

Целое в пределах от 0 до $2^{32} - 1$ *включительно* представляется 32-битным словом. Наименее значимые 4 бита целого представляются крайней правой 16-ричной цифрой слова. Например, целое $291 = 2^8 + 2^5 + 2^1 + 2^0 = 256 + 32 + 2 + 1$ представляется 16-ричным словом «00000123».

То же верно для целых в пределах от 0 до $2^{64} - 1$ *включительно*, которые представляются 64-битным словом.

Если Z – целое, и $0 \le Z < 2^{64}$, то $Z = 2^{32} X + Y$, где $0 \le X < 2^{32}$ и $0 \le Y < 2^{32}$. Так как X и Y могут быть представлены 32-битными словами x и y соответственно, то целое Z может быть представлено парой слов (x, y). Это свойство используется в SHA-1, SHA-224 и SHA-256.

Если Z – целое, и $0 \le Z < 2^{128}$, то $Z = 2^{64} X + Y$, где $0 \le X < 2^{64}$ и $0 \le Y < 2^{64}$. Так как X и Y могут быть представлены 64-битными словами x и y соответственно, то целое Z может быть представлено парой слов (x, y). Это свойство используется в SHA-384 и SHA-512.

4. Для алгоритмов безопасного хэша размер *блока сообщения (m бит)* зависит от алгоритма.

а) в **SHA-1**, **SHA-224** и **SHA-256** каждый блок сообщения имеет размер **512 бит** и представляется последовательностью из шестнадцати **32-битных слов**

б) в **SHA-384** и **SHA-512** каждый блок сообщения имеет размер **1024 бит** и представляется последовательностью из шестнадцати **64-битных слов**.

3.2 Операции над словами

Во всех пяти алгоритмах вычисления безопасного хэша к w-битным словам применяются перечисленные ниже операции. SHA-1, SHA-224 и SHA-256 оперируют 32-битными словами (w = 32). SHA-384 и SHA-512 оперируют 64-битными словами (w = 64).

1. побитовые логические операции над словом: $\wedge$, $\vee$, $\oplus$, $\neg$ (см. пункт 2.2.2).

2. сложение по модулю 2^w

Операция $x + y$ определяется следующим образом. Слова x и y представляют целые X и Y, где $0 \le X < 2^w$ и $0 \le Y < 2^w$. Для положительных целых U и V результатом операции $U \bmod V$ будет остаток от деления U на V. Вычисляем

$$Z = (X + Y) \bmod 2^w,$$

где $0 \le Z < 2^w$. Преобразуем целое Z в слово z и получим $z = x + y$.

3. операция сдвига вправо $SHR^n(x)$, где x – это w-битное слово, а n – это целое в пределах $0 \le n < w$, определяется как

$$SHR^n(x) = x >> n$$

Эта операция используется в алгоритмах SHA-224, SHA-256, SHA-384 и SHA-512.

4. операция циклического сдвига вправо $ROTR^n(x)$, где x – это w-битное слово, а n – это целое в пределах $0 \le n < w$, определяется как

$$ROTR^n(x) = (x >> n) \vee (x << w - n)$$

Таким образом, $ROTR^n(x)$ эквивалентно циклическому сдвигу x на n позиций вправо. Данная операция используется в алгоритмах SHA-224, SHA-256, SHA-384 и SHA-512.

5. Операция циклического сдвига влево $ROTL^n(x)$, где x – это w-битное слово, а n – это целое в пределах $0 \le n < w$, определяется как

$$ROTL^n(x) = (x << n) \vee (x >> w - n)$$

Таким образом, $ROTL^n(x)$ эквивалентно циклическому сдвигу x на n позиций влево. Эта операция используется только в алгоритме SHA-1.

6. Обратите внимание на следующие соотношения эквивалентности, где w фиксировано в каждом соотношении:

$$ROTL^n(x) \approx ROTR^{w-n}(x)$$
$$ROTR^n(x) \approx ROTL^{w-n}(x)$$

4. Функции и константы

4.1 Функции

В данном подразделе определяются функции, которые используются в каждом алгоритме. Хотя алгоритмы SHA-224, SHA-256, SHA-384 и SHA-512 используют одни и те же функции, описания этих функций разделены на два подраздела: подраздел 4.1.2 для SHA-224 и SHA-256 и подраздел 4.1.3 для SHA-384 и SHA-512. Это сделано из-за различий размера входных и выходных слов данных функций.

Каждый алгоритм использует функции *Ch*(*x, y, z*) и *Maj*(*x, y, z*). Замена в этих функциях операции исключающего ИЛИ ($\oplus$) на операцию побитового (видимо, хотели написать «включающего» - прим. пер.) ИЛИ ($\vee$) не приведёт к изменению выхода функции.

4.1.1 Функции SHA-1

SHA-1 использует последовательность логических функций $f_0, f_1, ..., f_{79}$. Каждая функция f_t, где $0 \le t < 79$ (видимо, хотели написать «≤ 79» – прим. пер.), выполняет операции над тремя 32-битными словами *x, y* и *z*. Выходом функции является 32-битное слово. Функция $f_t(x, y, z)$ определяется следующим образом:

$$f_t(x, y, z) = \begin{cases} Ch(x, y, z) = (x \wedge y) \oplus (\neg x \wedge z) & 0 \le t \le 19 \\ Parity(x, y, z) = x \oplus y \oplus z & 20 \le t \le 39 \\ Maj(x, y, z) = (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z) & 40 \le t \le 59 \\ Parity(x, y, z) = x \oplus y \oplus z & 60 \le t \le 79 \end{cases} \quad (4.1)$$

4.1.2 Функции SHA-224 и SHA-256

SHA-224 и SHA-256 используют шесть логических функций. *Каждая функция выполняет операции над 32-битными словами*, которые обозначены как *x, y* и *z*. Результатом каждой функции является новое 32-битное слово.

$$Ch(x, y, z) = (x \wedge y) \oplus (\neg x \wedge z) \quad (4.2)$$

$$Maj(x, y, z) = (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z) \quad (4.3)$$

$$\sum_0^{\{256\}}(x) = ROTR^2(x) \oplus ROTR^{13}(x) \oplus ROTR^{22}(x) \quad (4.4)$$

$$\sum_1^{\{256\}}(x) = ROTR^6(x) \oplus ROTR^{11}(x) \oplus ROTR^{25}(x) \quad (4.5)$$

$$s_0^{\{256\}}(x) = ROTR^7(x) \oplus ROTR^{18}(x) \oplus SHR^3(x) \quad (4.6)$$

$$s_1^{\{256\}}(x) = ROTR^{17}(x) \oplus ROTR^{19}(x) \oplus SHR^{10}(x) \quad (4.7)$$

4.1.3 Функции SHA-384 и SHA-512

SHA-384 и SHA-512 используют шесть логических функций. *Каждая функция выполняет операции над 64-битными словами*, которые обозначены как *x*, *y* и *z*. Результатом каждой функции является новое 64-битное слово.

$$Ch(x, y, z) = (x \wedge y) \oplus (\neg x \wedge z) \quad (4.8)$$

$$Maj(x, y, z) = (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z) \quad (4.9)$$

$$\sum_0^{\{512\}}(x) = ROTR^{28}(x) \oplus ROTR^{34}(x) \oplus ROTR^{39}(x) \quad (4.10)$$

$$\sum_1^{\{512\}}(x) = ROTR^{14}(x) \oplus ROTR^{18}(x) \oplus ROTR^{41}(x) \quad (4.11)$$

$$s_0^{\{512\}}(x) = ROTR^1(x) \oplus ROTR^8(x) \oplus SHR^7(x) \quad (4.12)$$

$$s_1^{\{512\}}(x) = ROTR^{19}(x) \oplus ROTR^{61}(x) \oplus SHR^6(x) \quad (4.13)$$

4.2 Константы

4.2.1 Константы SHA-1

SHA-1 использует последовательность из восьмидесяти констант $K_0, K_1, \mathbf{K}, K_{79}$. Константы являются 32-битными словами и определяются как:

$$K_t = \begin{cases} 5A827999 & 0 \le t \le 19 \\ 6ED9EBA1 & 20 \le t \le 39 \\ 8F1BBCDC & 40 \le t \le 59 \\ CA62C1D6 & 60 \le t \le 79 \end{cases} \quad (4.14)$$

4.2.2 Константы SHA-224 и SHA-256

SHA-224 и SHA-256 используют последовательность из 64 констант $K_0^{\{256\}}, K_1^{\{256\}}, \mathbf{K}, K_{63}^{\{256\}}$. Константы являются 32-битными словами. Эти слова представляют первые 32 бита дробных частей чисел, являющихся результатом вычисления кубических корней из первых 64 простых чисел. Константы в 16-ричном виде (слева на право):

```
428A2F98  71374491  B5C0FBCF  E9B5DBA5  3956C25B  59F111F1  923F82A4  AB1C5ED5
D807AA98  12835B01  243185BE  550C7DC3  72BE5D74  80DEB1FE  9BDC06A7  C19BF174
E49B69C1  EFBE4786  0FC19DC6  240CA1CC  2DE92C6F  4A7484AA  5CB0A9DC  76F988DA
983E5152  A831C66D  B00327C8  BF597FC7  C6E00BF3  D5A79147  06CA6351  14292967
27B70A85  2E1B2138  4D2C6DFC  53380D13  650A7354  766A0ABB  81C2C92E  92722C85
A2BFE8A1  A81A664B  C24B8B70  C76C51A3  D192E819  D6990624  F40E3585  106AA070
19A4C116  1E376C08  2748774C  34B0BCB5  391C0CB3  4ED8AA4A  5B9CCA4F  682E6FF3
748F82EE  78A5636F  84C87814  8CC70208  90BEFFFA  A4506CEB  BEF9A3F7  C67178F2
```

4.2.3 Константы SHA-384 и SHA-512

SHA-384 и SHA-512 используют последовательность из 80 констант $K_0^{\{512\}}, K_1^{\{512\}}, \mathbf{L}, K_{79}^{\{512\}}$. Константы являются 64-битными словами. Эти слова представляют первые 64 бита дробных частей чисел, являющихся результатом вычисления кубических корней из первых 80 простых чисел. Константы в 16-ричном виде (слева направо):

```
428A2F98D728AE22  7137449123EF65CD  B5C0FBCFEC4D3B2F  E9B5DBA58189DBBC
3956C25BF348B538  59F111F1B605D019  923F82A4AF194F9B  AB1C5ED5DA6D8118
D807AA98A3030242  12835B0145706FBE  243185BE4EE4B28C  550C7DC3D5FFB4E2
72BE5D74F27B896F  80DEB1FE3B1696B1  9BDC06A725C71235  C19BF174CF692694
E49B69C19EF14AD2  EFBE4786384F25E3  0FC19DC68B8CD5B5  240CA1CC77AC9C65
2DE92C6F592B0275  4A7484AA6EA6E483  5CB0A9DCBD41FBD4  76F988DA831153B5
983E5152EE66DFAB  A831C66D2DB43210  B00327C898FB213F  BF597FC7BEEF0EE4
C6E00BF33DA88FC2  D5A79147930AA725  06CA6351E003826F  142929670A0E6E70
27B70A8546D22FFC  2E1B21385C26C926  4D2C6DFC5AC42AED  53380D139D95B3DF
650A73548BAF63DE  766A0ABB3C77B2A8  81C2C92E47EDAEE6  92722C851482353B
A2BFE8A14CF10364  A81A664BBC423001  C24B8B70D0F89791  C76C51A30654BE30
D192E819D6EF5218  D69906245565A910  F40E35855771202A  106AA07032BBD1B8
19A4C116B8D2D0C8  1E376C085141AB53  2748774CDF8EEB99  34B0BCB5E19B48A8
391C0CB3C5C95A63  4ED8AA4AE3418ACB  5B9CCA4F7763E373  682E6FF3D6B2B8A3
748F82EE5DEFB2FC  78A5636F43172F60  84C87814A1F0AB72  8CC702081A6439EC
90BEFFFA23631E28  A4506CEBDE82BDE9  BEF9A3F7B2C67915  C67178F2E372532B
CA273ECEEA26619C  D186B8C721C0C207  EADA7DD6CDE0EB1E  F57D4F7FEE6ED178
06F067AA72176FBA  0A637DC5A2C898A6  113F9804BEF90DAE  1B710B35131C471B
28DB77F523047D84  32CAAB7B40C72493  3C9EBE0A15C9BEBC  431D67C49C100D4C
4CC5D4BECB3E42B6  597F299CFC657E2A  5FCB6FAB3AD6FAEC  6C44198C4A475817
```

5. Предварительная обработка

Предварительная обработка производится перед вычислением хэша. Она состоит из трёх действий: дополнения сообщения *M* (подраздел 5.1), разделения дополненного сообщения на блоки (подраздел 5.2) и задания начального значения хэша $H^{(0)}$ (подраздел 5.3).

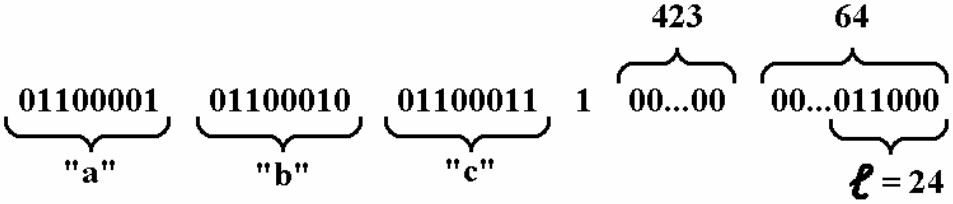
5.1 Дополнение сообщения

Перед началом вычисления хэша сообщение *M* должно быть дополнено. Дополнение выполняется с целью сделать длину дополненного сообщения кратной 512 или 1024 битам (в зависимости от алгоритма).

5.1.1 SHA-1, SHA-224 и SHA-256

Пусть длина сообщения *M* равна ℓ бит. В конец сообщения добавляется бит «1», затем k нулевых бит, где k – наименьшее неотрицательное решение уравнения $\ell + 1 + k \equiv 448 \bmod 512$. Затем добавляется блок из 64 бит, который является представлением числа ℓ в двоичном виде.

Например, сообщение «**abc**» (в 8-битном коде ASCII) имеет длину 8x3 = 24 бит. Таким образом, к сообщению сначала добавляется бит «1», затем 448 - (24 + 1) = 423 нулевых бита, затем длина сообщения. Получается дополненное сообщение длиной 512 бит.

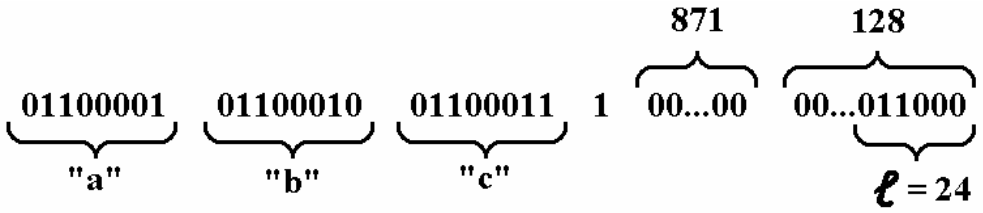


Длина дополненного сообщения должна быть кратна 512 битам.

5.1.2 SHA-384 и SHA-512

Пусть длина сообщения *M* равна ℓ бит. В конец сообщения добавляется бит «1», затем k нулевых бит, где k – наименьшее неотрицательное решение уравнения $\ell + 1 + k \equiv 896 \bmod 1024$. Затем добавляется блок из 128 бит, который является представлением числа ℓ в двоичном виде.

Например, сообщение «**abc**» (в 8-битном коде ASCII) имеет длину 8x3 = 24 бит. Таким образом, к сообщению сначала добавляется бит «1», затем 896 - (24 + 1) = 871 нулевых бит, затем длина сообщения. Получается дополненное сообщение длиной 1024 бит.



Длина дополненного сообщения должна быть кратна 1024 битам.

5.2 Разделение дополненного сообщения

После того как сообщение было дополнено, но перед вычислением хэша, оно должно быть разделено на N блоков из m бит.

5.2.1 SHA-1, SHA-224 и SHA-256

Для SHA-1, SHA-224 и SHA-256 дополненное сообщение разделяется на N блоков из 512 бит $M^{(1)}, M^{(2)}, \mathbf{K}, M^{(N)}$. Далее входной блок из 512 бит представляется как шестнадцать 32-битных слов. Первые 32 бита блока сообщения i обозначаются как $M_0^{(i)}$. Следующие 32 бита – как $M_1^{(i)}$ и так далее до $M_{15}^{(i)}$.

5.2.2 SHA-384 и SHA-512

Для SHA-384 и SHA-512 дополненное сообщение разделяется на N блоков из 1024 бит $M^{(1)}, M^{(2)}, \mathbf{K}, M^{(N)}$. Далее входной блок из 1024 бит представляется как шестнадцать 64-битных слов. Первые 64 бита блока сообщения i обозначаются как $M_0^{(i)}$. Следующие 64 бита – как $M_1^{(i)}$ и так далее до $M_{15}^{(i)}$.

5.3 Задание начального значения хэша ($H^{(0)}$)

Для каждого алгоритма перед началом вычисления хэша должно быть задано начальное значение хэша $H^{(0)}$. Разрядность и число слов, составляющих $H^{(0)}$, зависит от размера отпечатка сообщения.

5.3.1 SHA-1

Начальное значение хэша $H^{(0)}$ для SHA-1 состоит из следующих пяти 32-битных слов, представленных в 16-ричном виде:

$$H_0^{(0)} = 67452301$$
$$H_1^{(0)} = EFCDAB89$$
$$H_2^{(0)} = 98BADCFE$$
$$H_3^{(0)} = 10325476$$
$$H_4^{(0)} = C3D2E1F0$$

5.3.2 SHA-224

Начальное значение хэша $H^{(0)}$ для SHA-224 состоит из следующих восьми 32-битных слов, представленных в 16-ричном виде:

$$H_0^{(0)} = C1059ED8$$
$$H_1^{(0)} = 367CD507$$
$$H_2^{(0)} = 3070DD17$$
$$H_3^{(0)} = F70E5939$$
$$H_4^{(0)} = FFC00B31$$
$$H_5^{(0)} = 68581511$$
$$H_6^{(0)} = 64F98FA7$$
$$H_7^{(0)} = BEFA4FA4$$

5.3.3 SHA-256

Начальное значение хэша $H^{(0)}$ для SHA-256 состоит из следующих восьми 32-битных слов, представленных в 16-ричном виде:

$$H_0^{(0)} = \texttt{6A09E667}$$
$$H_1^{(0)} = \texttt{BB67AE85}$$
$$H_2^{(0)} = \texttt{3C6EF372}$$
$$H_3^{(0)} = \texttt{A54FF53A}$$
$$H_4^{(0)} = \texttt{510E527F}$$
$$H_5^{(0)} = \texttt{9B05688C}$$
$$H_6^{(0)} = \texttt{1F83D9AB}$$
$$H_7^{(0)} = \texttt{5BE0CD19}$$

Эти слова были получены путём взятия первых 32 бит от дробных частей чисел, являющихся результатом вычисления квадратных корней из первых 8 простых чисел.

5.3.4 SHA-384

Начальное значение хэша $H^{(0)}$ для SHA-384 состоит из следующих восьми 64-битных слов, представленных в 16-ричном виде:

$$H_0^{(0)} = \texttt{CBBB9D5DC1059ED8}$$
$$H_1^{(0)} = \texttt{629A292A367CD507}$$
$$H_2^{(0)} = \texttt{9159015A3070DD17}$$
$$H_3^{(0)} = \texttt{152FECD8F70E5939}$$
$$H_4^{(0)} = \texttt{67332667FFC00B31}$$
$$H_5^{(0)} = \texttt{8EB44A8768581511}$$
$$H_6^{(0)} = \texttt{DB0C2E0D64F98FA7}$$
$$H_7^{(0)} = \texttt{47B5481DBEFA4FA4}$$

Эти слова были получены путём взятия первых 64 бит от дробных частей чисел, являющихся результатом вычисления квадратных корней из простых чисел (начиная с девятого и заканчивая шестнадцатым).

5.3.5 SHA-512

Начальное значение хэша $H^{(0)}$ для SHA-512 состоит из следующих восьми 64-битных слов, представленных в 16-ричном виде:

$$H_0^{(0)} = \texttt{6A09E667F3BCC908}$$
$$H_1^{(0)} = \texttt{BB67AE8584CAA73B}$$
$$H_2^{(0)} = \texttt{3C6EF372FE94F82B}$$
$$H_3^{(0)} = \texttt{A54FF53A5F1D36F1}$$
$$H_4^{(0)} = \texttt{510E527FADE682D1}$$
$$H_5^{(0)} = \texttt{9B05688C2B3E6C1F}$$
$$H_6^{(0)} = \texttt{1F83D9ABFB41BD6B}$$
$$H_7^{(0)} = \texttt{5BE0CD19137E2179}$$

Эти слова были получены путём взятия первых 64 бит от дробных частей чисел, являющихся результатом вычисления квадратных корней из первых 8 простых чисел.

6. Алгоритмы вычисления безопасного хэша

В следующих подразделах алгоритмы вычисления хэша описаны не в порядке увеличения размера отпечатка. SHA-256 описан перед SHA-224, так как описания для SHA-224 и SHA-256

совпадают. Разница заключается в используемых начальных значениях и том, что в SHA-224 конечное значение хэша усекается до 224 бит.

То же верно для SHA-512 и SHA-384, только конечное значение хэша для SHA-384 усекается до 384 бит.

Для каждого алгоритма безопасного хэша могут существовать альтернативные способы вычисления, которые приведут к тем же результатам. В качестве примера в пункте 6.1.3 приведён альтернативный способ вычисления SHA-1.

Такие альтернативные способы могут быть реализованы и будут считаться соответствующими данному стандарту.

6.1 SHA-1

Алгоритм SHA-1 может быть использован для вычисления хэша сообщения ***M***, имеющего длину ℓ бит, где $0 \le \mathbf{l} < 2^{64}$. Алгоритм использует:

- последовательность блоков данных из восьмидесяти 32-битных слов
- пять рабочих переменных по 32 бита каждая
- значение хэша из пяти 32-битных слов.

Конечным результатом работы алгоритма SHA-1 является 160-битный отпечаток сообщения.

Слова, составляющие последовательность блоков данных, обозначены как $W_0, W_1, \mathbf{K}, W_{79}$. Пять рабочих переменных обозначены как ***a***, ***b***, ***c***, ***d*** и ***e***. Слова, составляющие значение хэша, обозначены как $H_0^{(i)}, H_1^{(i)}, \mathbf{K}, H_4^{(i)}$.

После обработки очередного блока предыдущее значение хэша (в самом начале равное $H^{(0)}$) будет заменяться на следующее промежуточное значение хэша $H^{(i)}$. Конечным значением хэша будет $H^{(N)}$.

Ещё SHA-1 использует одно временное слово ***T***.

6.1.1 Предварительная обработка SHA-1

1. Дополнить сообщение ***M*** в соответствии с пунктом 5.1.1.
2. Разделить дополненное сообщение на ***N*** блоков из 512 бит $M^{(1)}, M^{(2)}, \mathbf{K}, M^{(N)}$ в соответствии с пунктом 5.2.1.
3. Задать начальное значение хэша $H^{(0)}$ в соответствии с пунктом 5.3.1.

6.1.2 Вычисление хэша SHA-1

Алгоритм вычисления хэша SHA-1 использует функции и константы, ранее определённые в пунктах 4.1.1 и 4.2.1 соответственно. Сложение (+) производится по модулю 2^{32}.

После выполнения предварительной обработки каждый блок сообщения $M^{(1)}, M^{(2)}, \mathbf{K}, M^{(N)}$ по порядку обрабатывается в соответствии со следующими шагами:

для ***i*** начиная с 1 до ***N*** включительно выполнить:

{ 1. Подготовить последовательность блоков данных $\{W_t\}$:

$$W_t = \begin{cases} M_t^{(i)} & 0 \le t \le 15 \\ ROTL^1(W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16}) & 16 \le t \le 79 \end{cases}$$

2. Инициализировать пять рабочих переменных **a**, **b**, **c**, **d** и **e** (*i*-1)-м значением хэша:

$$a = H_0^{(i-1)}$$
$$b = H_1^{(i-1)}$$
$$c = H_2^{(i-1)}$$
$$d = H_3^{(i-1)}$$
$$e = H_4^{(i-1)}$$

3. Для t начиная с 0 до 79 включительно выполнить:

{

$T = ROTL^5(a) + f_t(b, c, d) + e + K_t + W_t$

$e = d$

$d = c$

$c = ROTL^{30}(b)$

$b = a$

$a = T$

}

4. Вычислить i-ое промежуточное значение хэша $H^{(i)}$:

$H_0^{(i)} = a + H_0^{(i-1)}$

$H_1^{(i)} = b + H_1^{(i-1)}$

$H_2^{(i)} = c + H_2^{(i-1)}$

$H_3^{(i)} = d + H_3^{(i-1)}$

$H_4^{(i)} = e + H_4^{(i-1)}$

}

После того как шаги с первого по четвёртый будут выполнены все N раз (то есть после обработки $M^{(N)}$), результирующим 160-битным отпечатком сообщения M будет:

$$H_0^{(N)} \parallel H_1^{(N)} \parallel H_2^{(N)} \parallel H_3^{(N)} \parallel H_4^{(N)}$$

6.1.3 Альтернативный способ вычисления отпечатка SHA-1

Способ вычисления хэша SHA-1, описанный в пункте 6.1.2, предполагает, что последовательность блоков данных $W_0, W_1, \mathbf{K}, W_{79}$ реализована как массив из восьмидесяти 32-битных слов. Это эффективно с точки зрения минимизации времени выполнения, так как адреса блоков $W_{t-3}, \mathbf{K}, W_{t-16}$ на шаге 2 (видимо, хотели написать «1» – прим. пер.) пункта 6.1.2 легко вычисляются.

Однако если количество памяти ограничено, то альтернативой является представление последовательности $\{W_t\}$ как круговой очереди, которая может быть реализована как массив из шестнадцати 32-битных слов $W_0, W_1, \mathbf{K}, W_{15}$.

Альтернативный способ, описанный в данном пункте, даёт точно такой же отпечаток, как и способ, описанный в пункте 6.1.2. Хотя данный альтернативный способ сохраняет в памяти шестьдесят четыре (видимо, хотели написать «шестнадцать» – прим. пер.) 32-битных числа, он, вероятно, увеличивает время выполнения вследствие увеличения сложности вычисления адреса для $\{W_t\}$ на шаге 3.

Для данного альтернативного способа вычисления SHA-1 число MASK = 0000000F (в 16-ричном виде). Как и в пункте 6.1.1 сложение выполняется по модулю 2^{32}. При условии, что предварительная обработка, описанная в пункте 6.1.1, уже выполнена, обработка $M^{(i)}$ происходит следующим образом:

для i начиная с 1 до N включительно выполнить

{ 1. Для t начиная с 0 до 15 включительно выполнить: $\{W_t = M_t^{(i)}\}$

2. Инициализировать пять рабочих переменных ***a***, ***b***, ***c***, ***d*** и ***e*** (i-1)-м значением хэша:

$a = H_0^{(i-1)}$

$b = H_1^{(i-1)}$

$c = H_2^{(i-1)}$

$d = H_3^{(i-1)}$

$e = H_4^{(i-1)}$

3. Для t начиная с 0 до 79 включительно выполнить:

{

$s = t \wedge MASK$

Если $t \geq 16$, то выполнить:

{

$W_s = ROTL^1(W_{(s+13)\wedge MASK} \oplus W_{(s+8)\wedge MASK} \oplus W_{(s+2)\wedge MASK} \oplus W_s)$

}

$T = ROTL^5(a) + f_t(b, c, d) + e + K_t + W_s$

$e = d$

$d = c$

$c = ROTL^{30}(b)$

$b = a$

$a = T$

}

4. Вычислить i-ое промежуточное значение хэша $H^{(i)}$:

$H_0^{(i)} = a + H_0^{(i-1)}$

$H_1^{(i)} = b + H_1^{(i-1)}$

$H_2^{(i)} = c + H_2^{(i-1)}$

$H_3^{(i)} = d + H_3^{(i-1)}$

$H_4^{(i)} = e + H_4^{(i-1)}$

}

После того как шаги с первого по четвёртый будут выполнены все N раз (то есть после обработки $M^{(N)}$), результирующим 160-битным отпечатком сообщения M будет:

$$H_0^{(N)} \parallel H_1^{(N)} \parallel H_2^{(N)} \parallel H_3^{(N)} \parallel H_4^{(N)}$$

6.2 SHA-256

Алгоритм SHA-256 может быть использован для вычисления хэша сообщения M длиной ℓ бит, где $0 \leq \mathbf{l} < 2^{64}$. Алгоритм использует:

- последовательность блоков данных из шестидесяти четырёх 32-битных слов
- восемь рабочих переменных по 32 бита каждая
- значение хэша из восьми 32-битных слов.

Конечным результатом работы алгоритма SHA-256 является 256-битный отпечаток сообщения.

Слова, составляющие последовательность блоков данных, обозначены как $W_0, W_1, \mathbf{K}, W_{63}$. Восемь рабочих переменных обозначены как ***a, b, c, d, e, f, g*** и ***h***. Слова, составляющие значение хэша, обозначены как $H_0^{(i)}, H_1^{(i)}, \mathbf{K}, H_7^{(i)}$.

После обработки очередного блока предыдущее значение хэша (в самом начале равное $H^{(0)}$) будет заменяться на следующее промежуточное значение хэша $H^{(i)}$. Конечным значением хэша будет $H^{(N)}$.

Ещё SHA-256 использует два временных слова T_1 и T_2.

6.2.1 Предварительная обработка SHA-256

1. Дополнить сообщение M в соответствии с пунктом 5.1.1.
2. Разделить дополненное сообщение на N блоков из 512-ти бит $M^{(1)}, M^{(2)}, \mathbf{K}, M^{(N)}$ в соответствии с пунктом 5.2.1.
3. Задать начальное значение хэша $H^{(0)}$ в соответствии с пунктом 5.3.3.

6.2.2 Вычисление хэша SHA-256

Алгоритм вычисления хэша SHA-256 использует функции и константы, ранее определённые в пунктах 4.1.2 и 4.2.2 соответственно. Сложение (+) производится по модулю 2^{32}.

После выполнения предварительной обработки каждый блок сообщения $M^{(1)}, M^{(2)}, \mathbf{K}, M^{(N)}$ по порядку обрабатывается в соответствии со следующими шагами:

для i начиная с 1 до N включительно выполнить

{ 1. Подготовить последовательность блоков данных $\{W_t\}$:

$$W_t = \begin{cases} M_t^{(i)} & 0 \le t \le 15 \\ s_1^{\{256\}}(W_{t-2}) + W_{t-7} + s_0^{\{256\}}(W_{t-15}) + W_{t-16} & 16 \le t \le 63 \end{cases}$$

2. Инициализировать восемь рабочих переменных ***a***, ***b***, ***c***, ***d***, ***e***, ***f***, ***g*** и ***h*** (i-1)-м значением хэша:

$$a = H_0^{(i-1)}$$
$$b = H_1^{(i-1)}$$
$$c = H_2^{(i-1)}$$
$$d = H_3^{(i-1)}$$
$$e = H_4^{(i-1)}$$
$$f = H_5^{(i-1)}$$
$$g = H_6^{(i-1)}$$
$$h = H_7^{(i-1)}$$

3. Для t начиная с 0 до 63 включительно выполнить:

{

$$T_1 = h + \sum_1^{\{256\}}(e) + Ch(e, f, g) + K_t^{\{256\}} + W_t$$
$$T_2 = \sum_0^{\{256\}}(a) + Maj(a, b, c)$$
$$h = g$$
$$g = f$$
$$f = e$$
$$e = d + T_1$$
$$d = c$$
$$c = b$$
$$b = a$$
$$a = T_1 + T_2$$

}

4. Вычислить i-ое промежуточное значение хэша $H^{(i)}$:

$$H_0^{(i)} = a + H_0^{(i-1)}$$
$$H_1^{(i)} = b + H_1^{(i-1)}$$
$$H_2^{(i)} = c + H_2^{(i-1)}$$
$$H_3^{(i)} = d + H_3^{(i-1)}$$
$$H_4^{(i)} = e + H_4^{(i-1)}$$
$$H_5^{(i)} = f + H_5^{(i-1)}$$
$$H_6^{(i)} = g + H_6^{(i-1)}$$
$$H_7^{(i)} = h + H_7^{(i-1)}$$

}

После того как шаги с первого по четвёртый будут выполнены все N раз (то есть после обработки $M^{(N)}$), результирующим 256-битным отпечатком сообщения M будет:

$$H_0^{(N)} \parallel H_1^{(N)} \parallel H_2^{(N)} \parallel H_3^{(N)} \parallel H_4^{(N)} \parallel H_5^{(N)} \parallel H_6^{(N)} \parallel H_7^{(N)}$$

6.3 SHA-224

Алгоритм SHA-224 может быть использован для вычисления хэша сообщения M длиной ℓ бит, где $0 \le \mathbf{l} < 2^{64}$. Функция определяется точно таким же образом, как и SHA-256 (подраздел 6.2), кроме следующих двух исключений:

1. Начальное значение хэша $H^{(0)}$ должно быть задано в соответствии с пунктом 5.3.2
2. 224-битный отпечаток получается путём усечения конечного значения хэша $H^{(N)}$, чтобы остались крайние левые 224 бита:

$$H_0^{(N)} \parallel H_1^{(N)} \parallel H_2^{(N)} \parallel H_3^{(N)} \parallel H_4^{(N)} \parallel H_5^{(N)} \parallel H_6^{(N)}$$

6.4 SHA-512

Алгоритм SHA-512 может быть использован для вычисления хэша сообщения M длиной ℓ бит, где $0 \le \mathbf{l} < 2^{128}$. Алгоритм использует:

- последовательность блоков данных из восьмидесяти 64-битных слов
- восемь рабочих переменных по 64 бита каждая
- значение хэша из восьми 64-битных слов.

Конечным результатом работы алгоритма SHA-512 является 512-битный отпечаток сообщения.

Слова, составляющие последовательность блоков данных, обозначены как $W_0, W_1, \mathbf{K}, W_{79}$.

Восемь рабочих переменных обозначены как ***a, b, c, d, e, f, g*** и ***h***. Слова, составляющие значение хэша, обозначены как $H_0^{(i)}, H_1^{(i)}, \mathbf{K}, H_7^{(i)}$.

После обработки очередного блока предыдущее значение хэша (в самом начале равное $H^{(0)}$) будет заменяться на следующее промежуточное значение хэша $H^{(i)}$. Конечным значением хэша будет $H^{(N)}$.

Ещё SHA-512 использует два временных слова T_1 и T_2.

6.4.1 Предварительная обработка SHA-512

1. Дополнить сообщение M в соответствии с пунктом 5.1.2.
2. Разделить дополненное сообщение на N блоков из 1024 бит $M^{(1)}, M^{(2)}, \mathbf{K}, M^{(N)}$ в соответствии с пунктом 5.2.2.
3. Задать начальное значение хэша $H^{(0)}$ в соответствии с пунктом 5.3.5.

6.4.2 Вычисление хэша SHA-512

Алгоритм вычисления хэша SHA-512 использует функции и константы, ранее определённые в пунктах 4.1.3 и 4.2.3 соответственно. Сложение (+) производится по модулю 2^{64}.

После выполнения предварительной обработки каждый блок сообщения $M^{(1)}, M^{(2)}, \mathbf{K}, M^{(N)}$ по порядку обрабатывается в соответствии со следующими шагами:

для i начиная с 1 до N включительно выполнить

{ 1. Подготовить последовательность блоков данных $\{W_t\}$:

$$W_t = \begin{cases} M_t^{(i)} & 0 \le t \le 15 \\ s_1^{\{512\}}(W_{t-2}) + W_{t-7} + s_0^{\{512\}}(W_{t-15}) + W_{t-16} & 16 \le t \le 79 \end{cases}$$

2. Инициализировать восемь рабочих переменных ***a***, ***b***, ***c***, ***d***, ***e***, ***f***, ***g*** и ***h*** (i-1)-м значением хэша:

$a = H_0^{(i-1)}$

$b = H_1^{(i-1)}$

$c = H_2^{(i-1)}$

$d = H_3^{(i-1)}$

$e = H_4^{(i-1)}$

$f = H_5^{(i-1)}$

$g = H_6^{(i-1)}$

$h = H_7^{(i-1)}$

3. Для t начиная с 0 до 79 включительно выполнить:

{

$T_1 = h + \sum_1^{\{512\}}(e) + Ch(e, f, g) + K_t^{\{512\}} + W_t$

$T_2 = \sum_0^{\{512\}}(a) + Maj(a, b, c)$

$h = g$

$g = f$

$f = e$

$e = d + T_1$

$d = c$

$c = b$

$b = a$

$a = T_1 + T_2$

}

4. Вычислить i-ое промежуточное значение хэша $H^{(i)}$:

$H_0^{(i)} = a + H_0^{(i-1)}$

$H_1^{(i)} = b + H_1^{(i-1)}$

$H_2^{(i)} = c + H_2^{(i-1)}$

$H_3^{(i)} = d + H_3^{(i-1)}$

$H_4^{(i)} = e + H_4^{(i-1)}$

$H_5^{(i)} = f + H_5^{(i-1)}$

$H_6^{(i)} = g + H_6^{(i-1)}$

$H_7^{(i)} = h + H_7^{(i-1)}$

}

После того как шаги с первого по четвёртый будут выполнены все N раз (то есть после обработки $M^{(N)}$), результирующим 512-битным отпечатком сообщения M будет:

$$H_0^{(N)} \parallel H_1^{(N)} \parallel H_2^{(N)} \parallel H_3^{(N)} \parallel H_4^{(N)} \parallel H_5^{(N)} \parallel H_6^{(N)} \parallel H_7^{(N)}$$

6.5 SHA-384

Алгоритм SHA-384 может быть использован для вычисления хэша сообщения M длиной ℓ бит, где $0 \le \mathbf{l} < 2^{128}$. Алгоритм определяется точно таким же образом, как и SHA-512 (подраздел 6.4), кроме следующих двух исключений:

1. Начальное значение хэша $H^{(0)}$ должно быть задано в соответствии с пунктом 5.3.4.

2. 384-битный отпечаток получается путём усечения конечного значения хэша $H^{(N)}$, чтобы остались крайние левые 384 бита:

$$H_0^{(N)} \parallel H_1^{(N)} \parallel H_2^{(N)} \parallel H_3^{(N)} \parallel H_4^{(N)} \parallel H_5^{(N)}$$

7. Усечение отпечатка

В некоторых задачах могут требоваться хэш-функции с длиной отпечатка, отличающейся от перечисленных в данном стандарте. В таких случаях может быть использовано усечение длины отпечатка. Хэш-функция с большей длиной отпечатка применяется к требуемым хэширования данным. Затем результирующий отпечаток усекается, чтобы осталось требуемое число крайних левых бит. Рекомендации по выбору длины усечённого отпечатка и информацию об их безопасности для приложений смотри в специальной публикации [SP 800-107].

Приложение А – дополнительная информация

А.1 Безопасность алгоритмов вычисления безопасного хэша

Безопасность пяти алгоритмов SHA-1, SHA-224, SHA-256, SHA-384 и SHA-512 обсуждается в [SP 800-107].

А.2 Замечания по реализации

Примеры вычисления SHA-1, SHA-224, SHA-256, SHA-384 и SHA-512 доступны по ссылке http://csrc.nist.gov/groups/ST/toolkit/examples.html.

А.3 Идентификаторы объектов

Идентификаторы объектов (OID-ы) для алгоритмов SHA-1, SHA-224, SHA-256, SHA-384 и SHA-512 доступны по ссылке http://csrc.nist.gov/groups/ST/crypto_apps_infra/csor/algorithms.html.

Приложение Б – ссылки

[FIPS 180-2] Национальный институт стандартов и технологий, федеральные стандарты обработки информации, публикация 180-2, *Стандарты безопасного хэша*, август 2001

[SP 800-57] Специальная публикация национального института стандартов и технологий (SP) 800-57, часть 1, *Рекомендация по управлению ключом: общее*, август 2005

[SP 800-107] Специальная публикация национального института стандартов и технологий (SP) 800-107, *Рекомендация для приложений, использующих утверждённые хэш-алгоритмы*, (черновая версия), июль 2008.