# Лекция 8. Хэш-функции

**Хэш функция** – преобразование, используемое для аутентификации сообщений

C**войства**:

- Len(a)-произвольная (применимость к блоку **данных** **любой длины**),  
- Len(H)=const (создание **выхода фиксированной длины**),  
- дешевизна(**легкость вычисления** для любого блока данных)  
- H(a)!<=a (**необратимость** - невозможность вычисления исходных данных по хэш)  
- a!=b => H(a)!=H(b)(невозможно найти два различных блока данных с одинаковым хэш)

Для гарантированного выявления изменений исходного сообщения используются **криптографические** хэш-функции.

Главная проблема – коллизия одинаковых хешей для разных данных (birthday атаки – вероятность нахождения в одном помещении людей с одинаковыми ДР).

Способ построения: генератор использует предыдущий блок (захешированный) и блок исходного сообщения. Это гарантирует ограничение результата заданным фиксированным размером.

Назначение

- Аутентификация (в памяти хранится хеш от пароля)  
- Защита данных (проверка целостности – с данными хранится и хеш)

- Разделение секрета (…)

Группы алгоритмов: MD(Message Digest), SHA(Secure Hash Algorithm)

MD5 и SHA-1 являются развитиями MD4 (в оба добавлен лавинный эффект).

Avalanche effect (лавинный эффект) - зависимость всех выходных битов от каждого входного бита

Пример плохого лавинного эффекта - [Шифр Цезаря](file:////wiki/%25D0%25A8%25D0%25B8%25D1%2584%25D1%2580_%25D0%25A6%25D0%25B5%25D0%25B7%25D0%25B0%25D1%2580%25D1%258F):

E(k = 3, "xxxxx") = aaaaa

E(k = 3, "xxYxx") = aaBaa

**Лавинный эффект порядка Х** (GAC - guaranteed avalanche criterion): при изменении 1 бита на входе на выходе меняются, как минимум, Х выходных битов.

Выполнение GAC порядка Х в диапазоне от 2 до 5 для узлов замен обеспечивает любому шифру очень высокий лавинный эффект вследствие распространения изменений в битах при прохождении данных по раундам шифрования в схеме Фейстеля

Лавинный эффект является следствием хорошей конфузии и диффузии.

**Диффузия** —избыточность в статистике входных данных «распределяется» по всей структуре выходных данных (от статистического анализа скрывается структура исходного текста).

Реализуется при помощи [P-блоков](file:////w/index.php%3ftitle=P-%25D0%25B1%25D0%25BB%25D0%25BE%25D0%25BA_(%25D0%25BA%25D1%2580%25D0%25B8%25D0%25BF%25D1%2582%25D0%25BE%25D0%25B3%25D1%2580%25D0%25B0%25D1%2584%25D0%25B8%25D1%258F)&action=edit&redlink=1) (каждый бит незашифрованного текста должен влиять на каждый бит зашифрованного текста).

**Конфузия** —зависимость ключа и выходных данных делается нелинейной.

Реализуется при помощи [S-блоков](file:////w/index.php%3ftitle=S-%25D0%25B1%25D0%25BB%25D0%25BE%25D0%25BA_(%25D0%25BA%25D1%2580%25D0%25B8%25D0%25BF%25D1%2582%25D0%25BE%25D0%25B3%25D1%2580%25D0%25B0%25D1%2584%25D0%25B8%25D1%258F)&action=edit&redlink=1).

**Лавинный эффект в DES**

**Подсчет зависимых выходных битов**

В DES лавинный эффект носит характер [**строгого лавинного эффекта**](file:////wiki/%25D0%259B%25D0%25B0%25D0%25B2%25D0%25B8%25D0%25BD%25D0%25BD%25D1%258B%25D0%25B9_%25D1%258D%25D1%2584%25D1%2584%25D0%25B5%25D0%25BA%25D1%2582#.D0.A1.D1.82.D1.80.D0.BE.D0.B3.D0.B8.D0.B9_.D0.BB.D0.B0.D0.B2.D0.B8.D0.BD.D0.BD.D1.8B.D0.B9_.D0.BA.D1.80.D0.B8.D1.82.D0.B5.D1.80.D0.B8.D0.B9) и проявляется уже на четвёртом-пятом раунде[[3]](#cite_note-StallingsCryptography-3), оценочно для каждой операции (считая, что к началу раунда влияние одного вначале выбранного бита распространилось на R 0 битов в правой части и L 0 — в левой):

После расширения: R 1 = min ( 1 , 5 ⋅ R 0 , 32 ) . Предполагая случайные попадания в 8 S-блоков, согласно [задаче о размещении](file:////w/index.php%3ftitle=%25D0%2597%25D0%25B0%25D0%25B4%25D0%25B0%25D1%2587%25D0%25B0_%25D0%25BE_%25D1%2580%25D0%25B0%25D0%25B7%25D0%25BC%25D0%25B5%25D1%2589%25D0%25B5%25D0%25BD%25D0%25B8%25D0%25B8&action=edit&redlink=1), биты попадут в: S 2 = 8 ⋅ ( 1 − ( 1 − 1 8 ) R 1 ) ≈ 8 ⋅ ( 1 − e − R 1 8 ) S-блоков.

Одно из требований [NSA](file:////wiki/%25D0%2590%25D0%25B3%25D0%25B5%25D0%25BD%25D1%2582%25D1%2581%25D1%2582%25D0%25B2%25D0%25BE_%25D0%25BD%25D0%25B0%25D1%2586%25D0%25B8%25D0%25BE%25D0%25BD%25D0%25B0%25D0%25BB%25D1%258C%25D0%25BD%25D0%25BE%25D0%25B9_%25D0%25B1%25D0%25B5%25D0%25B7%25D0%25BE%25D0%25BF%25D0%25B0%25D1%2581%25D0%25BD%25D0%25BE%25D1%2581%25D1%2582%25D0%25B8) к S-блокам заключалось в том, чтобы изменение каждого бита входа изменяло 2 бита выхода. Мы предположим, что каждый бит входа S-блока влияет на все 4 бита выхода. Зависимыми станут: S 2 = 4 ⋅ 8 ⋅ ( 1 − ( 1 − 1 8 ) R 1 ) ≈ 4 ⋅ 8 ⋅ ( 1 − e − R 1 8 ) битов.

После [битового сложения](file:////wiki/%25D0%25A1%25D0%25BB%25D0%25BE%25D0%25B6%25D0%25B5%25D0%25BD%25D0%25B8%25D0%25B5_%25D0%25BF%25D0%25BE_%25D0%25BC%25D0%25BE%25D0%25B4%25D1%2583%25D0%25BB%25D1%258E_2) левой L и правой R частей: R 3 ≈ R 2 + L 0 − R 2 ⋅ L 0 32 .

**Таблица распространения влияния 1 бита левой части в DES**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Раунд** | **L i** | **R i** | | |
| **l** | **Расширение r → R 1** | **S-блоки R 1 → R 2** | **R i + 1 = f ( R 1 ) ⊕ L i** |
| 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | (0, 1)→ 1 |
| 2 | 1 | 1 → 1,5 | 1,5 → 5,5 | (5,5, 0) → 5,5 |
| 3 | 5,5 | 5,5 → 8,3 | 8,2 → 20,5 | (20,5, 1) → 20,9 |
| 4 | 20,9 | 20,9 → 31.3 | 31,3 → 32 | (32, 20,9) → 32 |
| 5 | 32 | 32 | 32 | 32 |

Для достижения максимального лавинного эффекта требуется аккуратно выбрать расширение, S-блоки, а также перестановку в функции F

**Таблица зависимости количества измененных битов на каждом раунде**

Следующая таблица показывает количество измененных на каждом раунде выходных битов при условии изменения одного бита исходного текста и одного бита ключа. [[9]](#cite_note-Sourav-9)

|  |  |  |  |
| --- | --- | --- | --- |
| **Изменения во входном тексте** | | **Изменения в ключе** | |
| **Раунд** | **Количество измененных битов** | **Раунд** | **Количество измененных битов** |
| 0 | 1 | 0 | 0 |
| 1 | 6 | 1 | 2 |
| 2 | 21 | 2 | 14 |
| 3 | 35 | 3 | 28 |
| 4 | 39 | 4 | 32 |
| 5 | 34 | 5 | 30 |
| 6 | 32 | 6 | 32 |
| 7 | 31 | 7 | 35 |
| 8 | 29 | 8 | 34 |
| 9 | 42 | 9 | 40 |
| 10 | 44 | 10 | 38 |
| 11 | 32 | 11 | 31 |
| 12 | 30 | 12 | 33 |
| 13 | 30 | 13 | 28 |
| 14 | 26 | 14 | 26 |
| 15 | 29 | 15 | 34 |
| 16 | 34 | 16 | 35 |

**Лавинный эффект в AES**

В AES лавинный эффект появляется следующим образом: в первом раунде операция MixColumns() распространяет изменения одного байта на все 4 байта колонки, после чего во втором раунде применение операций ShiftRows() и MixColumns() распространяет изменения на всю таблицу. Таким образом, полная диффузия достигается уже на втором раунде. Конфузия обеспечивается операцией SubBytes().

Тест:

В таблице приведены численные значения лавинного эффекта для S-блоков в AES. В первом случае биты[*уточнить*] «11» меняются на «10», затем «11» меняется на «12», «00» на «10». Соответственно посчитан коэффициент лавинного эффекта для каждого случая. По этим данным мы можем наглядно видеть, что зашифрованный выходной текст совсем не простой, при довольно простом входном тексте[[10]](#cite_note-Kumar-10). А коэффициент лавинного эффекта близок к 0,5, что означает хорошее выполнение лавинного критерия.

|  |  |  |
| --- | --- | --- |
| **Входной текст** | **Зашифрованный текст[*уточнить*]** | **Лавинный эффект** |
| 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 | 79 F8 CC 24 01 82 DD 7F 2D 89 F7 E7 78 B7 EE 30 | 0,4375(56) |
| 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 10 | 9D 4C 1D B4 6A 93 27 B5 20 64 37 D1 3D 9D 2A |
| 11 22 33 66 55 44 55 44 77 88 99 66 44 45 36 12 | 4A A9 16 11 E2 8A 9F 67 35 30 1F 80 16 C5 B7 CD | 0,5153(66) |
| 11 22 33 66 55 44 55 44 77 88 99 66 44 45 36 11 | D7 00 43 2D 51 78 F7 65 50 03 03 75 B1 E4 2D A0 |
| 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | C6 A1 3B 37 87 8F 5B 82 6F 4F 81 62 A1 C8 79 | 0,4453(57) |
| 10 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | 0D 19 33 06 27 42 FE 01 9C FE 06 E1 A8 1A A0 01 |

Пример AES (изменение одного бита в данных или в ключе - > изменение шифра):

AES(ключ = "aaaaaaaaaaaaaaaa", "aaaaaaaaaaaaaaaa") = '5188c6474b228cbdd242e9125ebe1d53'

AES(ключ = "aaaaaaaaaaaaaaaa", "aa**c**aaaaaaaaaaaaa") = 'f7e5c1118f5cb86198e37ff7a29974bc'

AES(ключ = "aa**c**aaaaaaaaaaaaa", "aaaaaaaaaaaaaaaa") = '2c50b5cac9c755d67aa61b87c26248eb'

AES(ключ = "aa**c**aaaaaaaaaaaaa", "aa**c**aaaaaaaaaaaaa") = '87c09128de852b990b3dfbd65c7d9094'

**Лавинный эффект в ГОСТ 28147-89**

Основная статья: [**ГОСТ 28147-89**](file:////wiki/%25D0%2593%25D0%259E%25D0%25A1%25D0%25A2_28147-89)

Лавинный эффект по входу обеспечивается (4 на 4) S-блоками и циклическим сдвигом влево на 11 ≠ 0 mod 4

**Таблица распространения влияния 1 бита левой части в ГОСТ 28147-89**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Раунд** | **Li** | | | | | | | | **Ri** | | | | | | | |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 0 |  |  |  |  |  |  |  | 1 |  |  |  |  |  |  |  |  |
| 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 1 |
| 2 |  |  |  |  |  |  |  | 1 |  |  |  |  | 3 | 1 |  |  |
| 4 |  | 3 | 4 | 1 |  |  |  | 1 | 4 | 1 |  |  | 3 | 1 | 3 | 4 |
| 5 | 4 | 1 |  |  | 3 | 1 | 3 | 4 |  | 3 | 4 | 4 | 4 | 4 | 4 | 1 |
| 6 |  | 3 | 4 | 4 | 4 | 4 | 4 | 1 | 4 | 4 | 4 | 4 | 4 | 3 | 3 | 4 |
| 7 | 4 | 4 | 4 | 4 | 4 | 3 | 3 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 |
| 8 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 |

Из таблицы видно, что на каждом раунде число независимых битов увеличивается в среднем в 4 раза в результате сдвига и попадания выхода S-блока предыдущего раунда в 2 S-блока следующего раунда. Показано распространение зависимых битов в группах по 4 бита в левой и правой частях без учета сложения с ключом раунда. Предполагается, что каждый бит на входе S-блока влияет на все биты выхода. Число раундов для достижения полного лавинного эффекта без учета сложения с ключом — 8. Экспериментальная проверка для S-блоков, используемых [Центробанком РФ](file:////wiki/%25D0%25A6%25D0%25B5%25D0%25BD%25D1%2582%25D1%2580%25D0%25B0%25D0%25BB%25D1%258C%25D0%25BD%25D1%258B%25D0%25B9_%25D0%25B1%25D0%25B0%25D0%25BD%25D0%25BA_%25D0%25A0%25D0%25BE%25D1%2581%25D1%2581%25D0%25B8%25D0%25B9%25D1%2581%25D0%25BA%25D0%25BE%25D0%25B9_%25D0%25A4%25D0%25B5%25D0%25B4%25D0%25B5%25D1%2580%25D0%25B0%25D1%2586%25D0%25B8%25D0%25B8), показывает, что требуется 8 раундов[[*источник?*](file:////wiki/%25D0%2592%25D0%25B8%25D0%25BA%25D0%25B8%25D0%25BF%25D0%25B5%25D0%25B4%25D0%25B8%25D1%258F:%25D0%25A1%25D1%2581%25D1%258B%25D0%25BB%25D0%25BA%25D0%25B8_%25D0%25BD%25D0%25B0_%25D0%25B8%25D1%2581%25D1%2582%25D0%25BE%25D1%2587%25D0%25BD%25D0%25B8%25D0%25BA%25D0%25B8)].

**Значение лавинного эффекта в ГОСТ 28147-89**

Для криптостойкости ключевыми требованиями к операциям преобразования битов в раунде шифрования являются нелинейность, то есть невозможность подобрать линейную функцию, хорошо аппроксимирующую данное преобразование, и лавинный эффект — выполнение этих требований затрудняет проведение [линейного](file:////wiki/%25D0%259B%25D0%25B8%25D0%25BD%25D0%25B5%25D0%25B9%25D0%25BD%25D1%258B%25D0%25B9_%25D0%25BA%25D1%2580%25D0%25B8%25D0%25BF%25D1%2582%25D0%25BE%25D0%25B0%25D0%25BD%25D0%25B0%25D0%25BB%25D0%25B8%25D0%25B7) и [дифференциального](file:////wiki/%25D0%2594%25D0%25B8%25D1%2584%25D1%2584%25D0%25B5%25D1%2580%25D0%25B5%25D0%25BD%25D1%2586%25D0%25B8%25D0%25B0%25D0%25BB%25D1%258C%25D0%25BD%25D1%258B%25D0%25B9_%25D0%25BA%25D1%2580%25D0%25B8%25D0%25BF%25D1%2582%25D0%25BE%25D0%25B0%25D0%25BD%25D0%25B0%25D0%25BB%25D0%25B8%25D0%25B7) криптоанализа шифра соответственно. Если рассмотреть с этих позиций операции преобразования в раунде шифрования по ГОСТ 28147-89, то легко убедиться в том, что криптостойкость обеспечивают лишь операции сложения с ключом и выполнения замены битов по таблице, так как операции побитового сдвига и суммирования по модулю 2 являются линейными и не обладают лавинным эффектом. Из этого можно сделать вывод, что определяющим фактором надежности шифрования по ГОСТ 28147-89 является надлежащим образом выбранная ключевая информация (ключ и таблица замен). В случае зашифрования данных с нулевым ключом и тривиальной таблицей замен, все узлы которой содержат числа от 0 до 15 в порядке возрастания, найти по известному шифртексту открытый текст достаточно просто при помощи как линейного, так и дифференциального криптоанализа.

Как показано[[11]](#cite_note-Charnes-11), операция сложения данных с подключом не может обеспечить достаточного лавинного эффекта, поскольку при изменении одного бита на входе этой процедуры лишь один бит на выходе меняется с вероятностью 0,5, остальные биты меняются с вероятностью существенно меньшей. Это говорит о том, что для обеспечения криптостойкости шифрования недостаточно только обеспечения достаточного качества ключа — необходимо также использовать сильные таблицы замен с высокими показателями нелинейности и лавинного эффекта[[12]](#cite_note-12).

**MD5** (RFC 1321)

Вход (данные) – блоки по 128 b

Выход (хэш) - 160 b – мало!

1991 - Разработан профессором Рональдом Л. Ривестом из MIT (Massachusetts Institute of Technology), улучшенная версия MD4.

1993 - Берт ден Боер (Bert den Boer) и Антон Босселарис (Antoon Bosselaers) нашли в алгоритме коллизии (разным **инициализирующим векторам** соответствуют одинаковые дайджесты)

1996 - Ганс Доббертин (Hans Dobbertin) объявил о коллизии в **алгоритме**

март 2004- запущен проект MD5CRK для обнаружения уязвимостей алгоритма, используя birthday атаки (MD5CRK закончился 17 августа 2004 года, когда Ван Сяоюнь (Wang Xiaoyun), Фен Дэнгуо (Feng Dengguo), Лай Сюэцзя (Lai Xuejia) и Юй Хунбо (Yu Hongbo) обнаружили уязвимости в алгоритме).

1 марта 2005 - Arjen Lenstra, Xiaoyun Wang и Benne de Weger продемонстрировали построение двух X.509 документов с различными открытыми ключами и одинаковым хешем MD5.

18 марта 2006 - Властимил Клима (Vlastimil Klima) опубликовал алгоритм, находящий коллизии за 1 минуту на обычном компьютере (метод «туннелирование»).

Пример MD5:

MD5(0110 0001 0110 00**0**1 0110 0001 0110 00**0**1) = '74b87337454200d4d33f80c4663dc5e5'

MD5(0110 0001 0110 00**1**1 0110 0001 0110 00**0**1) = 'ca7de9e17429612452a717a44c36e688'

MD5(0110 0001 0110 00**0**1 0110 0001 0110 00**1**1) = '3963a2ba65ac8eb1c6e2140460031925'

**SHA-0** (RFC 3174)

Вход (данные) – блоки по 512 b

Выход (хэш) – 160 b

1993- разработан NSA и NIST.

Вскоре NSA отозвало данную версию, сославшись на обнаруженную ими ошибку, которая так и не была раскрыта.

1998 - на конференции CRYPTO два французских исследователя представили атаку на алгоритм SHA-0

**SHA-1**

Вход (данные) - блоки по 512 b

Выход (хеш) – 160 b

1995 - опубликован в FIPS PUB 180-1

Рекомендован в качестве основного для государственных учреждений в США.

Длина сообщения < 2^64

Алгоритм SHA-1:

1. Выбирается 5 констант (ABCDE) по 32 b
2. Блок 512 b разбиваются на 16 блоков по 32 b
3. Из них получаем 80 блоков по 32b (первые 16 копируются, остальные получаются xor из комбинации предыдущих 4 и других преобразований)
4. 80 раундов (or, xor, and для блоков) и в конце xor с 5 константами ABCDE
5. Получаем новые 5 блоков (константы для следующего шага, на последнем блоке - результат)

Пример SHA-1:

SHA-1(0110 0001 0110 00**0**1 0110 0001 0110 00**0**1) = '70c881d4a26984ddce795f6f71817c9cf4480e79'

SHA-1(0110 0001 0110 00**1**1 0110 0001 0110 00**0**1) = 'c6fdc1a445881de1f33e09cf00420a57b493b96d'

SHA-1(0110 0001 0110 00**0**1 0110 0001 0110 00**1**1) = '00b25f15212ed225d3389b5f75369c82084b3a76'

**SHA-2**

Собирательное название однонаправленных хеш-функций SHA-224, SHA-256, SHA-384 и SHA-512.

Вход (данные) - блоки по 512 или 1024 b   
Выход (хеш)– 256/224 или 512/384 b

Август 2002 - опубликован в FIPS PUB 180-2

В стандарт вошла хеш-функция SHA-1, разработанная в 1995 году.

февраль 2004 - в FIPS PUB 180-2 добавлена SHA-224.

июль 2006 - стандарт RFC 4634 «Безопасные хеш-алгоритмы США (SHA и HMAC-SHA)», описывающий SHA-1 и SHA-2.

октябрь 2008 - новая редакция стандарта FIPS PUB 180-3.

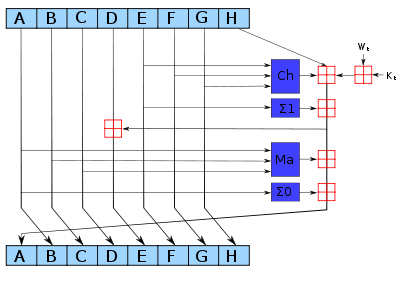
март 2012 - ?

АНБ от лица государства выпустило патент на SHA-2 под лицензией Royalty Free.

Хеш-функции SHA-2 построены на основе [структуры Меркла-Дамгарда](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%B0_%D0%9C%D0%B5%D1%80%D0%BA%D0%BB%D0%B0%E2%80%94%D0%94%D0%B0%D0%BC%D0%B3%D0%B0%D1%80%D0%B4%D0%B0).

Алгоритм SHA-2:

1. Сообщение дополняется до (кратности размеру блока) и разбивается на блоки по 512/1024 b
2. Выбирается 8 констант
3. Блок (512/1024 b) делятся на 16 слов (по 32/64 b)
4. 64/80 раундов для каждого блока: 2 слова преобразуются, функцию преобразования задают остальные слова
   1. 48 слов из исходных 16
   2. цикл по 64 словам



1. Получаем новые 8 переменных h0-h7, добавив (xor) к ним результат циклов
2. Следующий блок инициализируется результатом обработки предыдущего блока ( не получится независимо обрабатывать блоки и складывать результаты)
3. Результат= конкатенация результатов обработки каждого блока.

Алгоритм использует следующие битовые операции:

•ǁ — конкатенация,

•+ — сложение,

•and — побитовое «И»,

•or — побитовое «ИЛИ»,

•xor — исключающее «ИЛИ»,

•shr (shift right) — логический сдвиг вправо,

•rotr (rotate right) — циклический сдвиг вправо.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Хеш-функция** | **Длина дайджеста сообщения (бит)** | **Длина внутреннего состояния (бит)** | **Длина блока (бит)** | **Максимальная длина сообщения (бит)** | **Длина слова (бит)** | **Количество итераций в цикле** |
| SHA‑256, SHA‑224 | 256/224 | 256 | 512 | 264 − 1 | 32 | 64 |
| SHA‑512, SHA‑384, SHA‑512/256, SHA‑512/224 | 512/384/256/224 | 512 | 1024 | 2128 − 1 | 64 | 80 |

Псевдокод SHA-256

*Пояснения:*

Все переменные беззнаковые, имеют размер 32 бита и при вычислениях суммируются по модулю 232

*message* — исходное двоичное сообщение

*m* — преобразованное сообщение

*Инициализация переменных*

(первые 32 бита дробных частей квадратных корней первых восьми простых чисел [от 2 до 19]):

h0 := 0x6A09E667

h1 := 0xBB67AE85

h2 := 0x3C6EF372

h3 := 0xA54FF53A

h4 := 0x510E527F

h5 := 0x9B05688C

h6 := 0x1F83D9AB

h7 := 0x5BE0CD19

*Таблица констант*

(первые 32 бита дробных частей кубических корней первых 64 простых чисел [от 2 до 311]):

k[0..63] :=

0x428A2F98, 0x71374491, 0xB5C0FBCF, 0xE9B5DBA5, 0x3956C25B, 0x59F111F1, 0x923F82A4, 0xAB1C5ED5,

0xD807AA98, 0x12835B01, 0x243185BE, 0x550C7DC3, 0x72BE5D74, 0x80DEB1FE, 0x9BDC06A7, 0xC19BF174,

0xE49B69C1, 0xEFBE4786, 0x0FC19DC6, 0x240CA1CC, 0x2DE92C6F, 0x4A7484AA, 0x5CB0A9DC, 0x76F988DA,

0x983E5152, 0xA831C66D, 0xB00327C8, 0xBF597FC7, 0xC6E00BF3, 0xD5A79147, 0x06CA6351, 0x14292967,

0x27B70A85, 0x2E1B2138, 0x4D2C6DFC, 0x53380D13, 0x650A7354, 0x766A0ABB, 0x81C2C92E, 0x92722C85,

0xA2BFE8A1, 0xA81A664B, 0xC24B8B70, 0xC76C51A3, 0xD192E819, 0xD6990624, 0xF40E3585, 0x106AA070,

0x19A4C116, 0x1E376C08, 0x2748774C, 0x34B0BCB5, 0x391C0CB3, 0x4ED8AA4A, 0x5B9CCA4F, 0x682E6FF3,

0x748F82EE, 0x78A5636F, 0x84C87814, 0x8CC70208, 0x90BEFFFA, 0xA4506CEB, 0xBEF9A3F7, 0xC67178F2

*Предварительная обработка:*

m := message ǁ [*единичный бит*]

m := m ǁ [k *нулевых бит*], где **k** — наименьшее неотрицательное число, такое что

(L + 1 + K) mod 512 = 448, где L — число бит в сообщении ([сравнима по модулю](https://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B8%D0%BC%D0%BE%D1%81%D1%82%D1%8C_%D0%BF%D0%BE_%D0%BC%D0%BE%D0%B4%D1%83%D0%BB%D1%8E) 512 c 448)

m := m ǁ *Длина*(message) — длина исходного сообщения в битах в виде 64-битного числа

с [порядком байтов](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D1%80%D1%8F%D0%B4%D0%BE%D0%BA_%D0%B1%D0%B0%D0%B9%D1%82%D0%BE%D0%B2) от старшего к младшему

*Далее сообщение обрабатывается последовательными порциями по 512 бит:*

разбить сообщение на куски по 512 бит

**для** каждого куска

разбить кусок на 16 слов длиной 32 бита (с [порядком байтов](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D1%80%D1%8F%D0%B4%D0%BE%D0%BA_%D0%B1%D0%B0%D0%B9%D1%82%D0%BE%D0%B2) от старшего к младшему внутри слова): w[0..15]

*Сгенерировать дополнительные 48 слов:*

**для** i **от** 16 **до** 63

s0 := (w[i-15] **rotr** 7) **xor** (w[i-15] **rotr** 18) **xor** (w[i-15] **shr** 3)

s1 := (w[i-2] **rotr** 17) **xor** (w[i-2] **rotr** 19) **xor** (w[i-2] **shr** 10)

w[i] := w[i-16] **+** s0 **+** w[i-7] **+** s1

*Инициализация вспомогательных переменных:*

a := h0

b := h1

c := h2

d := h3

e := h4

f := h5

g := h6

h := h7

*Основной цикл:*

**для** i **от** 0 **до** 63

Σ0 := (a **rotr** 2) **xor** (a **rotr** 13) **xor** (a **rotr** 22)

Ma := (a **and** b) **xor** (a **and** c) **xor** (b **and** c)

t2 := Σ0 + Ma

Σ1 := (e **rotr** 6) **xor** (e **rotr** 11) **xor** (e **rotr** 25)

Ch := (e **and** f) **xor** ((**not** e) **and** g)

t1 := h + Σ1 + Ch + k[i] + w[i]

h := g

g := f

f := e

e := d + t1

d := c

c := b

b := a

a := t1 + t2

*Добавить полученные значения к ранее вычисленному результату:*

h0 := h0 + a

h1 := h1 + b

h2 := h2 + c

h3 := h3 + d

h4 := h4 + e

h5 := h5 + f

h6 := h6 + g

h7 := h7 + h

*Получить итоговое значения хеша:*

digest = hash = h0 ǁ h1 ǁ h2 ǁ h3 ǁ h4 ǁ h5 ǁ h6 ǁ h7

Отличия SHA-224 от SHA-256:

- для инициализации переменных h0—h7 используются другие начальные значения,

- итоговый хеш **не учитывает** **h7** (H=h0||…||h6 - получаем 224 b вместо 256 b).

*Начальные значения переменных h0—h7 в SHA-224:*

h0 := 0xC1059ED8

h1 := 0x367CD507

h2 := 0x3070DD17

h3 := 0xF70E5939

h4 := 0xFFC00B31

h5 := 0x68581511

h6 := 0x64F98FA7

h7 := 0xBEFA4FA4

Отличия SHA-512:

- 80 раундов вместо 64,

- переменные и константы (длина слова) 64 b вместо 32 b,

- сдвиг в операциях rotr и shr производится на другое число позиций.

Отличия SHA-384от SHA-512:

- переменные h0—h7 имеют другие начальные значения,

- итоговый хеш **не использует h6 и h7**.

*Начальные значения переменных h0—h7 в SHA-384*

(первые 64 бита дробных частей квадратных корней простых чисел с 9-го по 16-е [от 23 до 53]):

h0 := CBBB9D5DC1059ED8

h1 := 629A292A367CD507

h2 := 9159015A3070DD17

h3 := 152FECD8F70E5939

h4 := 67332667FFC00B31

h5 := 8EB44A8768581511

h6 := DB0C2E0D64F98FA7

h7 := 47B5481DBEFA4FA4

SHA-512/256 идентичен SHA-512, за исключением:

- переменные h0—h7 имеют другие начальные значения,

- итоговый хеш обрезается до левых 256 бит (Н=h0||h1||h2||h3).

*Начальные значения переменных h0—h7 в SHA-512/256*:

h0 := 22312194FC2BF72C

h1 := 9F555FA3C84C64C2

h2 := 2393B86B6F53B151

h3 := 963877195940EABD

h4 := 96283EE2A88EFFE3

h5 := BE5E1E2553863992

h6 := 2B0199FC2C85B8AA

h7 := 0EB72DDC81C52CA2

SHA-512/224 идентичен SHA-512, за исключением:

- переменные h0—h7 имеют другие начальные значения,

- итоговый хеш обрезается до левых 224 бит (хеш=3,5 первых h)

*Начальные значения переменных h0—h7 в SHA-512/224*:

h0 := 8C3D37C819544DA2

h1 := 73E1996689DCD4D6

h2 := 1DFAB7AE32FF9C82

h3 := 679DD514582F9FCF

h4 := 0F6D2B697BD44DA8

h5 := 77E36F7304C48942

h6 := 3F9D85A86A1D36C8

h7 := 1112E6AD91D692A1