Лекция 6 БСит

Представьте себе ситуацию: вам отправили по электронной почте документ с конфиденциальной информацией по финансированию на будущий год. Вам необходима абсолютная уверенность в том, что полученный файл совершенно идентичен оригиналу и содержащиеся в нем цифры не были изменены «в пути». Подозрение, что документ был подделан, появляется, если некоторые цифры не сходятся, а электронная передача велась через внешнюю почтовую систему. Для этого существуют алгоритмы идентификации цифровой информации. Такие алгоритмы работают только с цифровой информацией. При необходимости идентификации текстового документи или изображения, нужно с помощью кодирования вначале получить цифровой аналог данной информации.

Различают несколько способов идентификации: контрольные суммы, контроль CRC, хэширование и цифровая подпись – таковы базовые средства аутентификации при цифровой передаче данных.

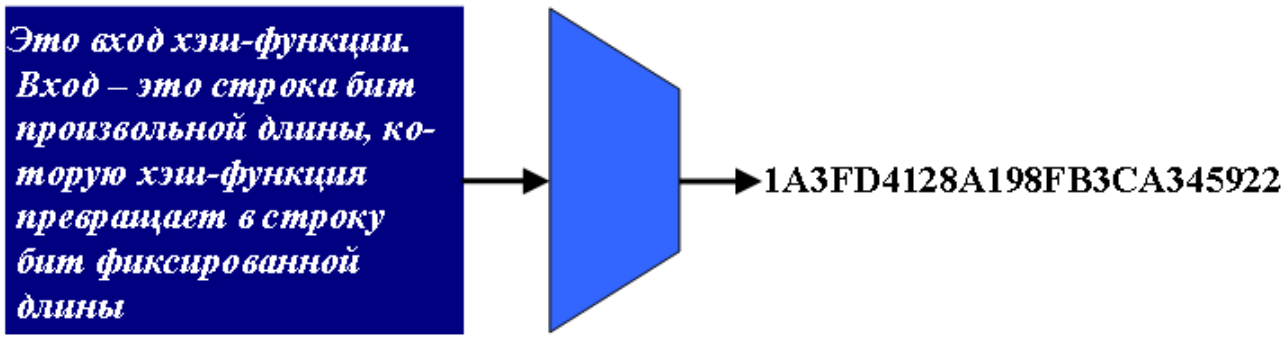
Контрольные суммы – наиболее простой способ проверки целостности данных, передаваемых в цифровом виде, при котором вычисляется контрольная сумма сообщения (определенное значение, которое и идентифицирует цифровую информацию).

Контроль CRC – более надежный способ цифровой идентификации данных. Это вычисление контрольного значения циклического избыточного кода данных (cyclic redundancy check – CRC) .

Наиболее совершенный способ идентификации цифровой информации – алгоритмы цифровой подписи и хеширование.

§ 1. КРИПТОГРАФИЧЕСКИЕ ХЭШ-ФУНКЦИИ

Функция хэширования $H(m)$ или **хэш-функция** (hash-function) – это детерминированная функция, на вход которой подается строка битов произвольной длины, а выходом всегда является битовая строка фиксированной длины n.



Значение хэш-функции $H(m)$ для входа m называют ***хэшем***.

В литературе можно широко распространены и другие названия, а именно

Хэш = ***хэш-образ= хэш-код*** = ***свертка =***
=дайджест сообщения =
криптографическая контрольная сумма =
= цифровой от печаток =
= код аутентичности сообщения =
= код обнаружения манипуляций.

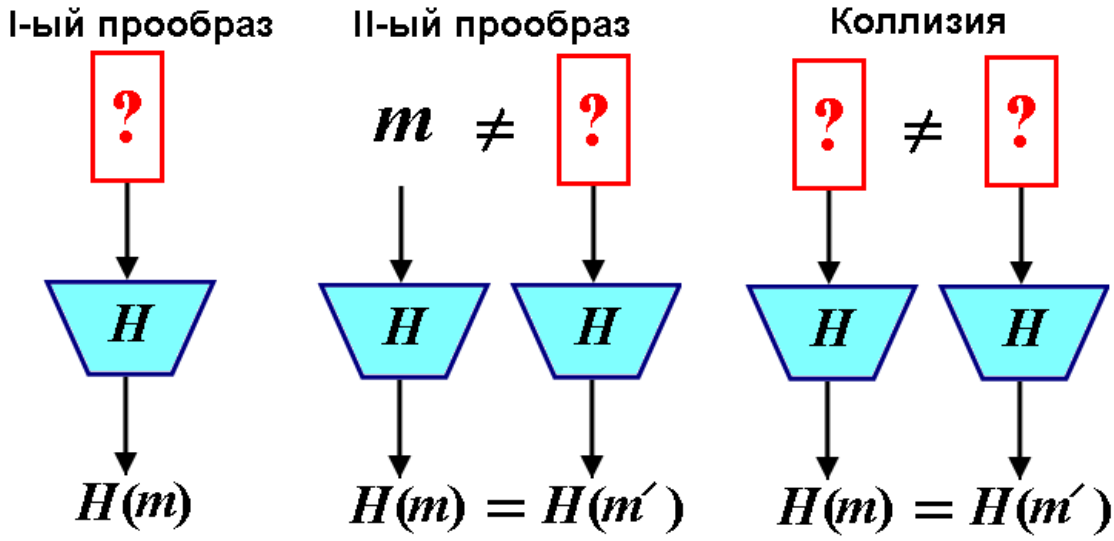
Исходная строка m, для которой вычислено хэш-значение, называется ***прообразом*** хэш-функции.

Свойства, которые должны быть присущи криптографическим хэш-функциям:

1. ***Стойкость к поиску первого прообраза*** – отсутствие эффективного полиномиального алгоритма вычисления обратной функции, т.е. нельзя восстановить текст m по известной его свертке $H(m)$ за реальное время (***необратимость***). Это свойство эквивалентно тому, что хэш-функция является односторонней функцией.

2. ***Стойкость к поиску второго прообраза (коллизиям первого рода)*** – вычислительно невозможно, зная сообщение m и его свертку $H(m)$, найти такое другое сообщение $m' \neq m$, чтобы $H(m) = H(m')$.

3. ***Стойкость к коллизиям (коллизиям второго рода)*** –. ***Коллизией*** для хэш-функции называется такая пара значений m и m', $m \neq m'$, для которой $H(m) = H(m')$. Так как количество возможных открытых текстов больше числа возможных значений свертки, то для некоторой свертки найдется много прообразов $\Rightarrow$ коллизии для хэш-функций обязательно существуют. Например, пусть длина хэш-прообраза 6 битов, длина свертки 4 бита. Тогда число различных



сверток – $2^4 = 16$, а число хэш-прообразов – $2^6 = 64$, т.е. в 4 раза

больше $\Rightarrow$ хотя бы одна свертка из всех соответствует 4 прообразам. ***Стойкость хэш-функции к коллизиям*** означает, что нет эффективного полиномиального алгоритма, позволяющего находить коллизии.

Замечание: свойства не являются независимыми:

- Обратимая функция неустойчива к восстановлению второго прообраза и коллизиям.
- Функция, нестойкая к восстановлению второго прообраза, нестойка к коллизиям; обратное неверно.
- Функция устойчивая к коллизиям, устойчива к нахождению второго прообраза.
- Устойчивая к коллизиям хэш-функция не обязательно является односторонней.

Для криптографии важно, чтобы значения хэш-функции сильно изменялись при малейшем изменении аргумента, т.е. ей должен быть присущ ***лавинный эффект***.

Значение хеша не должно давать утечки информации даже об отдельных битах аргумента.

§ 2. ПРИМЕНЕНИЕ ХЕШ-ФУНКЦИЙ ДЛЯ ПРОВЕРКИ ИСТИННОСТИ СООБЩЕНИЙ

Основные шаги процесса хэширования:

- Отправитель **A** подает сообщение на вход функции хэширования и находит его свертку (хэш).
- Свертка сообщения добавляется к сообщению.
- Отправитель **A** отправляет получателю сообщение+свертку .
- Получатель пропускает сообщение через функцию хэширования.
- Получатель **B** генерирует свое собственное значение свертки сообщения и сравнивает две свертки сообщения. Если они совпадают, сообщение не было изменено.

Возможная атака: криптоаналитик, перехватив сообщение, изменяет его, рассчитывает новый хэш, присоединяет его к измененному сообщению и отправляет получателю **B**. Когда тот получит такое сообщение, он проверит хеш, но так и не узнает о подделке.

Для более высокого уровня защиты, надо использовать код аутентификации сообщений (MAC – Message Authentication Code).

Вопросы:

1. Можно ли использовать обычный метод сжатия без потерь, например ZIP, в роли криптографической хэш-функции?

   Ответ: нет, сжатие без потерь создает сжатое сообщение, которое должно быть обратимо.
2. Можно ли использовать функцию контрольной суммы как криптографическую хэш-функцию?

Ответ: Нет, функция контрольной суммы дает не стойкий прообраз, так как можно найти несколько сообщений, контрольная сумма которых одинакова.

3. Хэш-функция $H(m) = m(\mathrm{mod}\, n)$, $H(m_1)$ и $H(m_2)$ – свертки сообщений m_1 и m_2. Можно ли выбрать в роли значения свертки сообщения $m_3 = m_1 + m_2$ значение $(H(m_1) + H(m_2))(\mathrm{mod}\, n)$?

   Ответ: Нет, нарушается свойство перемешивания.

4. Будет ли стойкая к коллизиям функция обязательно односторонней?

   Ответ: нет, например, функция $H(M) = M$ стойкая к коллизиям и определению второго прообраза, но не односторонняя.

§3. СХЕМА МЕРКЕЛЯ – ДАМГАРДА

На практике вместо создания хэш-функций, на вход которой можно подавать сообщения разной длины, используют необходимое количество раз функцию с входом фиксированного размера, называемую ***одношаговой функцией сжатия***. Функция сжатия – это функция двух переменных $y = f(x_1, x_2)$, где x_1, x_2 – битовые строки длины m, y – битовая строка длины n (n – длина свертки). Наиболее распространенной криптографической функцией хэширования на основе одношаговой функции сжатия является итеративная схема Меркеля – Дамгарда.

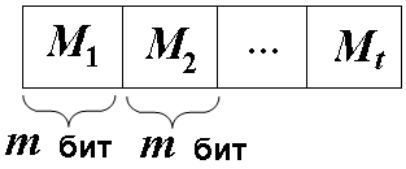
Схема Меркеля – Дамгарда

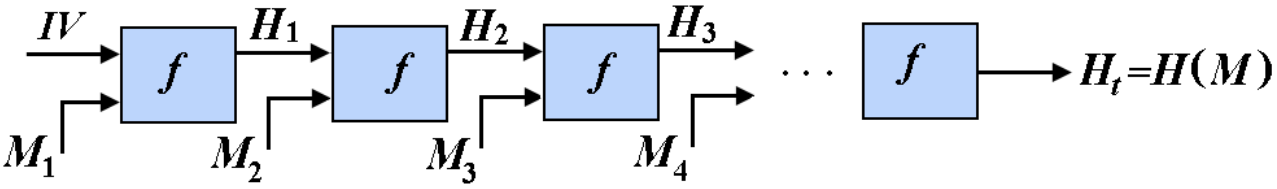
В конец хэшируемого сообщения M приписывается длина сообщения и дополнение, так чтобы длина увеличенного сообщения была кратна числу m, где m – длина строки, которую может обработать функция сжатия. Пусть сообщение разбито на t m-битовых блоков (строк): $M_1, M_2, ..., M_t$. Свертки, получаемые в ходе итераций, обозначим $H_1, H_2, ..., H_t$, результирующая свертка всего сообщения $H(M)$. Последовательная процедура вычисления свертки:

$$H_0 = IV;$$
$$H_i = f(M_i, H_{i-1}),\ i = 1, ..., t;$$
$$H(M) = H_t,$$

где IV – начальное значение свертки (фиксированный вектор), f – функция сжатия. Получаемая по такой схеме хэш-функция называется ***итерированной***.

Доказано, что если функция сжатия устойчива к коллизии, то хэш-функция также устойчива к коллизии.





§ 4. ДВА ТИПА КРИПТОГРАФИЧЕСКИХ ХЭШ-ФУНКЦИЙ

Типы криптографических хэш-функций

Ключевые хеш-функции или ***коды аутентификации сообщений*** (***Message Authentication Code - MAC***) Используются в системах с доверяющими друг другу пользователями

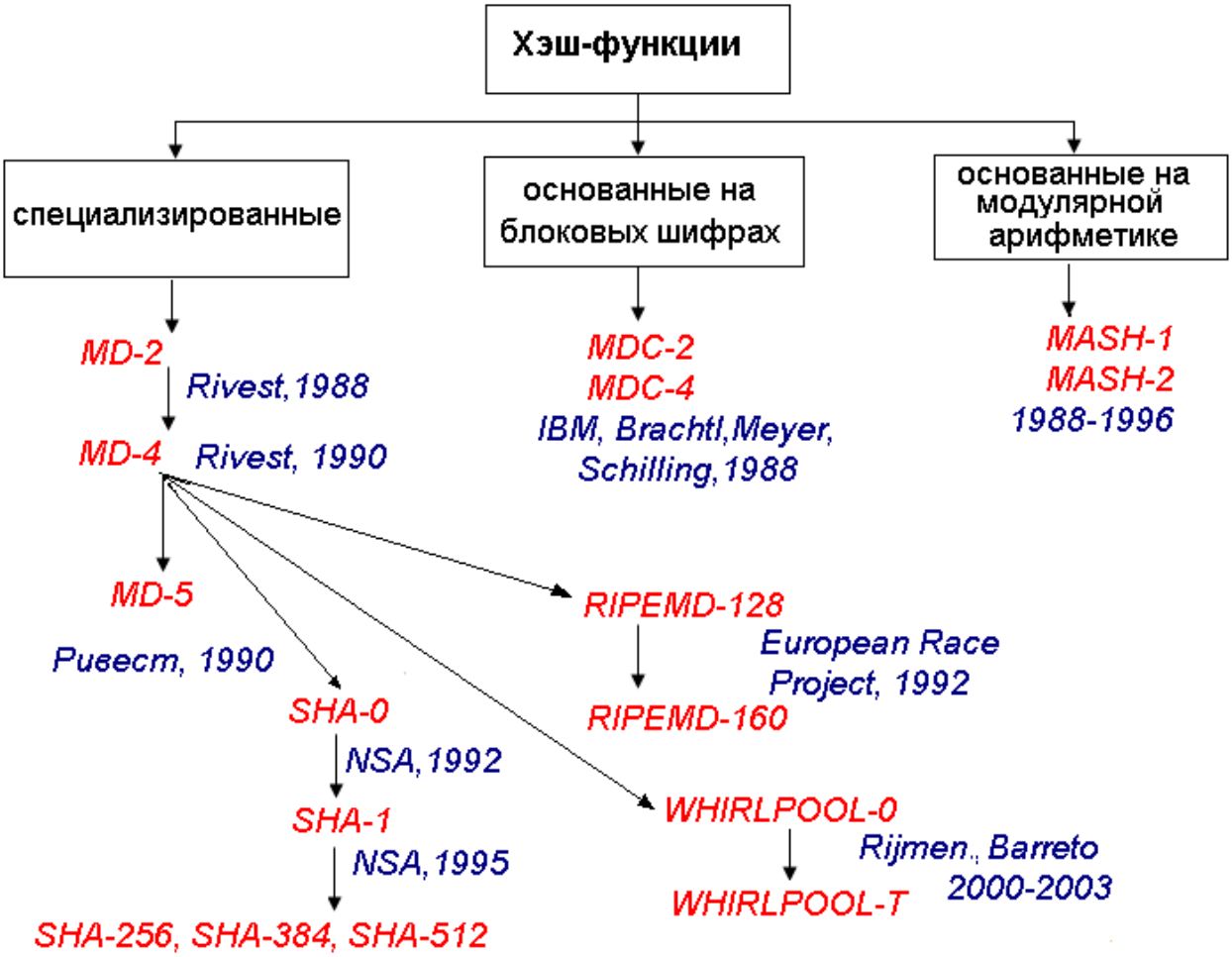
Бесключевые хеш-функции или ***коды обнаружения ошибок*** (***Modification Code – MDC*** или ***Message Integrity Code –MIC***) Используются в системах как с доверяющими друг другу, так и с недоверяющими пользователями

Разработка хэш-функций, удовлетворяющих всем требованиям, – сложная задача. Практически отвечают этим требованиям хэш-функции из группы алгоритмов MD и SHA.

Алгоритм	Описание	Длина хэша, бит	Примечание
MD2	Одностороння функция	128	Медленнее, чем MD4 и MD5
MD4	Одностороння функция	128	
MD5	Одностороння функция	128	Сложнее, но медленнее, чем MD4.
HAVAL	Одностороння функция	Длина различна	Модификация MD5, но более стойкая к атакам
SHA	Одностороння	160	Используется в

	функция		DSA
SHA-1, SHA-256, SHA-384, SHA-512	Обновленная версия SHA	SHA-1 создает хэш длиной 160 бит, SHA-256 – длиной 256 бит и т.д.	

Алгоритмы группы MD разработаны Роном Ривестом. Название обозначает «дайджест сообщения». ***Алгоритм безопасного хэширования*** (SHA – Secure Hash Algorithm) – стандарт, разработанный NIST и принятый как федеральный стандарт обработки информации в США. SHA основан на схеме Меркеля-Дамгарда.



§ 5. АЛГОРИТМ БЕЗОПАСНОГО ХЭШИРОВАНИЯ SHA-1

Основные характеристики алгоритма:

Длина хэш-кода – 160 бит

Длина обрабатываемых блоков – 512 бит

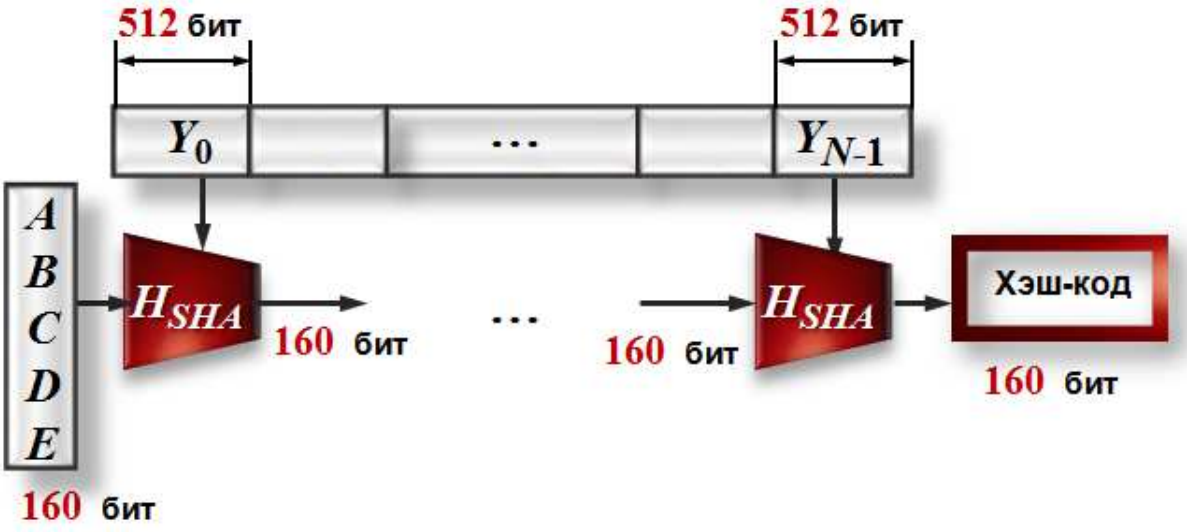
Число шагов алгоритма - 80 (4 раунда по 20 шагов)

Максимальная длина хэшируемых данных – $2^{64} - 1$.

1. Логика выполнения SHA-1

Алгоритм получает на входе сообщение максимальной длины $2^{64}-1$ бит и создает в качестве выхода дайджест сообщения длиной 160 бит.

Алгоритм состоит из следующих шагов:



Логика выполнения SHA-1

2. Добавление недостающих битов и указание длины (шаг 1)

Исходное m-битовое сообщение $(a_1, a_2, ..., a_m)$ расширяется до длины, кратной 512, по принципу:

$$(a_1, a_2, ..., a_m, \underbrace{1,0,0...,0}_{s-1}, l_1, l_2, ..., l_{64}),$$

где $(l_1, l_2, ..., l_{64})$ – двоичная запись числа m, s – наименьшее число, при котором $(m+s+64) \vdots 512$.

Пример 1. На вход алгоритма хеширования SHA-1 подается сообщение длиной 2590 битов. Сколько битов будет содержать дополнение сообщения?

Р е ш е н и е. $m+s+64 \equiv 0 (\mathrm{mod}\, 512) \Rightarrow s = -m-64 (\mathrm{mod}\, 512)$

$s = -2590 - 64 (\mathrm{mod}\, 512) = -2654 (\mathrm{mod}\, 512) \equiv 418.$

Дополнение состоит из одной 1 и 417 нулей.

Пример 2. Нужно ли добавлять дополнение, если длина первоначального сообщения уже кратна 512 битам?

Ответ: Да, так как надо еще приписать длину m исходного сообщения. Дополнение необходимо, чтобы сделать и новый блок кратным 512 битам.

Пример 3. Какое минимальное и максимальное число битов дополнения можно добавить к сообщению?

Ответ: Минимальная длина дополнения – 0, это в случае когда $-m-64(\mathrm{mod}\,512)\equiv 0$, тогда $m=-64\equiv 448(\mathrm{mod}\,512)$. Максимальная длина заполнения – 511 битов, когда $-m-64(\mathrm{mod}\,512)\equiv 511(\mathrm{mod}\,512)$ и $m=449(\mathrm{mod}\,512)$.

Итак, результатом первого шага является сообщение, длина которого кратна 512 битам. Далее расширенное сообщение делится на 512-битные блоки $Y_0, Y_1, ..., Y_{N-1}$, поэтому общая длина расширенного сообщения есть $N\cdot 512$ бит. Этот результат кратен шестнадцати 32-битным словам.

3. Инициализация буфера (шаг 2)

В алгоритме используется 160-битный буфер для хранения промежуточных и окончательных результатов хэш-функции. Буфер может быть представлен как пять 32-битных регистров A, B, C, D, E. Эти регистры инициализируются следующими шестнадцатеричными числами:

$$A = 67\ 45\ 23\ 01;$$
$$B = EF\ CD\ AB\ 89;$$
$$C = 98\ BA\ DC\ FE;$$
$$D = 10\ 32\ 54\ 76;$$
$$E = C3\ D2\ E1\ F0.$$

Вектор инициализации, подаваемый на вход 1-го раунда – результат конкатенации

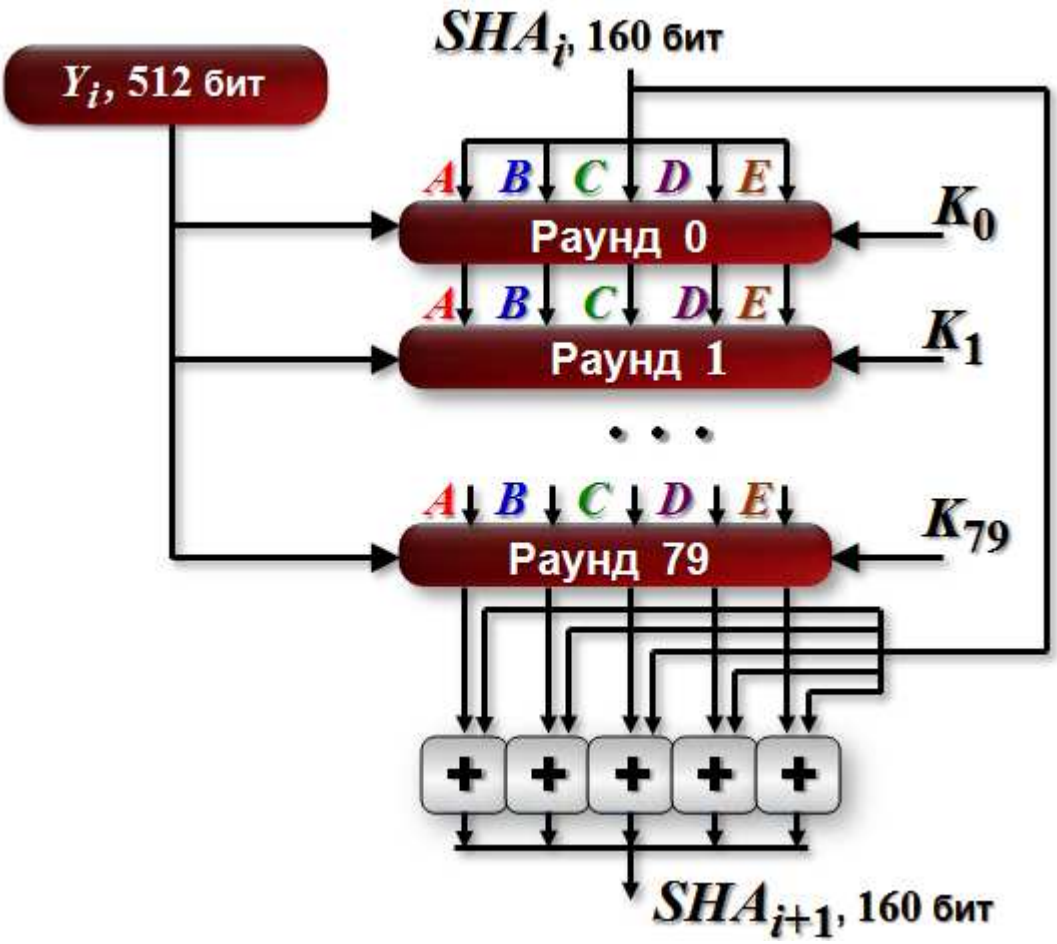
$$SHA_0 = A \,||\, B \,||\, C \,||\, D \,||\, E.$$

4. Обработка сообщения в 512-битных блоках (шаг 3)

Основой алгоритма является модуль, состоящий из 80 циклических обработок, обозначенный как H_{SHA}. Все 80 циклических обработок имеют одинаковую структуру.

Каждый цикл получает на входе текущий 512-битный обрабатываемый блок Y_i и 160-битное значение буфера $ABCDE$, и изменяет содержимое этого буфера.

В каждом цикле используется дополнительная константа K_t, которая принимает только четыре различных значения:



Обработка очередного 512-битного блока

$$K_t = \begin{cases} 5A827999, & 0 \le t \le 19; \\ 6ED9EBA1, & 20 \le t \le 39\ , \\ 8F1BBCDC, & 40 \le t \le 59; \\ CA62C1D6, & 60 \le t \le 79. \end{cases}$$

Для получения SHA_{i+1} выход 80-го цикла складывается со значением SHA_i. Сложение по модулю 2^{32} выполняется независимо для каждого из пяти слов в буфере с каждым из соответствующих слов в SHA_i.

5. Выход (шаг 4)

После обработки всех 512-битных блоков выходом является 160-битный дайджест сообщения.

6. Описание раунда обработки одного 512-битного блока даннях

Рассмотрим более детально логику в каждом из 80 раундов обработки одного 512-битного блока. Каждый раунд можно представить в виде:

$$TEMP(A,B,C,D,E,K_t,W_t) = S^5(A) + f_t(B,C,D) + E + K_t + W_t;$$

$$E = D;\ \ D = C;\ C = S^{30}(B);\ \ B = A;\ A = TEMP,$$

где t – номер раунда;

$S^5(A)$ – циклический левый сдвиг 32-битового аргумента A на 5 бит;

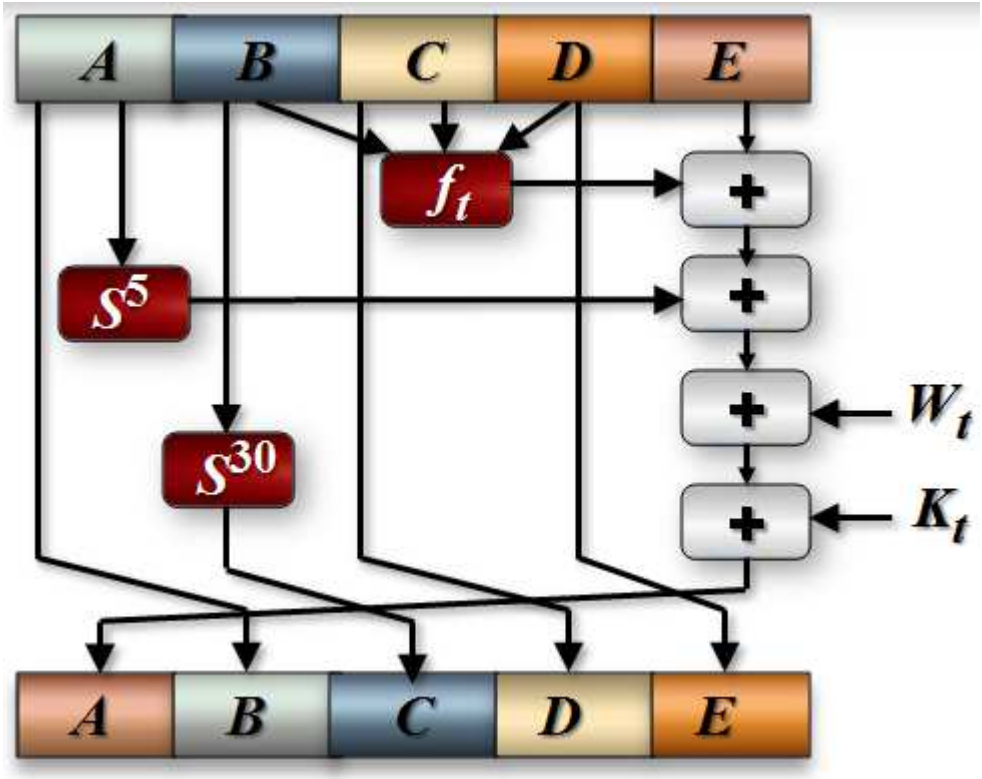
$S^{30}(B)$ – циклический левый сдвиг 32-битового аргумента B на 30 бит;

символ $+$ означает сложение по модулю 2^{32};

f_t – элементарная логическая функция (описана ниже);

K_t – пошаговые константы, равные

W_t –32-битные слова, получаемые из очередного 512-битного блока сообщения (способ генерации слов приведен ниже).



Логика работы одного раунда

Замечание: значения пошаговых констант K_t в алгоритме были выбраны по принципу ($[x]$– целая часть числа)

$$5A827999 = [2^{30} \cdot \sqrt{2}],$$
$$6ED9EBA1 = [2^{30} \cdot \sqrt{3}],$$
$$8F1BBCDC = [2^{30} \cdot \sqrt{5}],$$
$$CA62C1D6 = [2^{30} \cdot \sqrt{10}].$$

6. Логические функции f_t, используемые в раунде

Каждая элементарная функция получает на входе три 32-битовых слова и создает на выходе одно 32-битовое слово. Элементарная функция выполняет набор побитовых логических операций (побитовое умножение и сложение), т.е. n-ый бит выхода является функцией от n-ых битов трех входов. Функции следующие (t — номер раунда):

$$f_t(B,C,D) = \begin{cases} (B \wedge C) \vee (\overline{B} \wedge D), & 0 \le t \le 19; \\ B \oplus C \oplus D, & 20 \le t \le 39,\ 60 \le t \le 79; \\ (B \wedge C) \vee (B \cdot D) \vee (C \wedge D), & 40 \le t \le 59. \end{cases}$$

($\overline{B}$ получено из B инвертированием бит).

Пример. Крайние левые шестнадцатеричные цифры переменных B,C,D равны 07_h, $0A_h$ и $0E_h$ соответственно. Какие цифры будут крайними левыми в значении функции $f_{43}(B,C,D)$, используемой в алгоритме SHA?

Р е ш е н и е. $07_h = 0111_2$, $0A_h = 1010_2$, $0E_h = 1110_2$.

$(B \wedge C) \vee (B \wedge D) \vee (C \wedge D) = (0 \wedge 1) \vee (0 \wedge 1) \vee (1 \wedge 1) = 0 \vee 0 \vee 1 = 1.$

$(1 \wedge 0) \vee (1 \wedge 1) \vee (0 \wedge 1) = 0 \vee 1 \vee 0 = 1;$

$(1 \wedge 1) \vee (1 \wedge 1) \vee (1 \wedge 1) = 1 \vee 1 \vee 1 = 1;$

$(1 \wedge 0) \vee (1 \wedge 0) \vee (0 \wedge 0) = 0 \vee 0 \vee 0 = 0.$

Результат $1110_2 = 0E_h$.

7. Формирование входных значений раунда из очередного блока

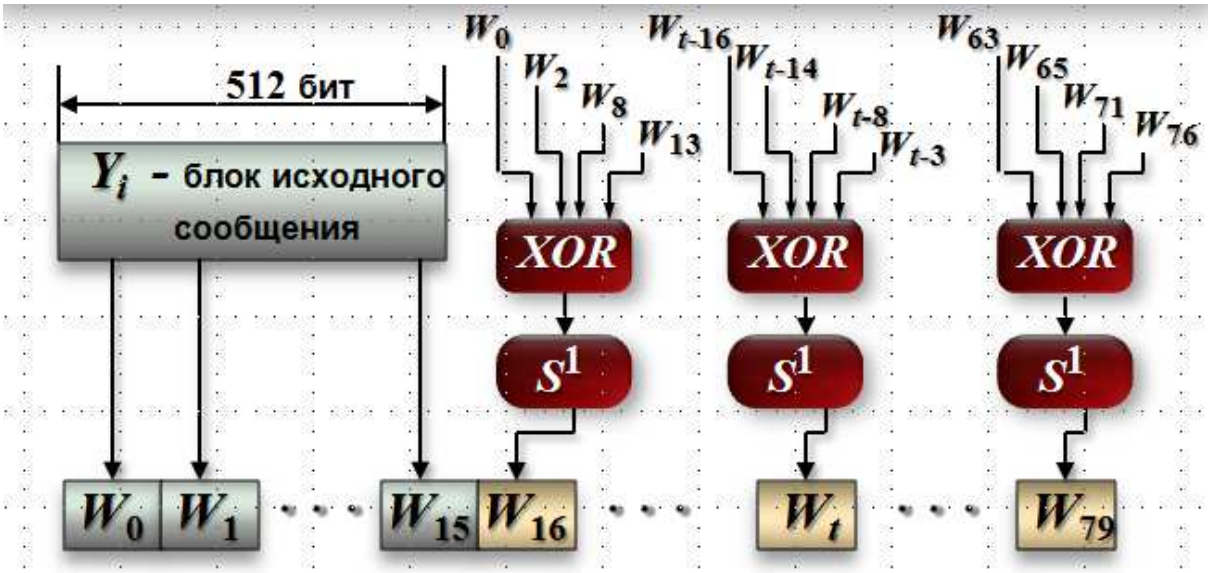
Перед началом первого раунда блок расширяют до 80 слов по 32 бита. Для этого вначале сам входной блок данных делят на 16 слов

$Y_0, Y_1, ..., Y_{15}$ по 32 бита и далее к ним присоединяют еще 64 слова по 32 бита. В результате образуются слова: $W_0, W_1, ..., W_{79}$. Принцип образования новых слов:

$$W_t = Y_t, \quad \text{при } t = 0,1,...,15;$$

$$W_t = S^1(W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16}) \text{ при } t = 16,...,79,$$

где S^1 – операция циклического сдвига аргумента на один разряд влево.



Алгоритм *SHA-1* можно суммировать следующим образом:

$$SHA_0 = IV;$$

$$SHA_{i+1} = \sum 32(SHA_i, ABCDE_i);$$

$$SHA = SHA_{N-1},$$

где

IV - начальное значение буфера $ABCDE$;

$ABCDE_i$ - результат обработки *i*-того блока сообщения;

N - число блоков в сообщении, включая поля добавления и длины;

$\sum 32$ - сумма по модулю 2^{32}, выполняемая отдельно для каждого слова буфера;

SHA - значение дайджеста сообщения.

(*Способ быстрого подбора коллизий для HAVAL, RIPEMD и MD5 нашли китайские исследователи, выложив в Сети файлы, которых до этого официально не мог видеть ни один человек на Земле (ну разве что тайно) – два разных файла, создающих одинаковое хэш-значение алгоритмом MD5! Удивленные пользователи скачивали файлы с примерами и убеждались, что это все правда. На подбор коллизии по "китайскому методу" (его подробности, кстати, китайцы не хотят публиковать целиком) уходит от 15 секунд до пары часов на персональном компьютере!*).

Упражнения

1. Каково дополнение для *SHA-512*, если длина сообщения:
   - 5120 битов
   - 5121 бит
   - 6143 бита
2. Найдите результат функции $f_{47}(B,C,D)$, если

   B = 1234 5678 ABCD 2345 34564 5678 ABCD 2468
   C = 2234 5678 ABCD 2345 34564 5678 ABCD 2468
   D = 3234 5678 ABCD 2345 34564 5678 ABCD 2468
3. Найдите результат функции $f_{13}(B,C,D)$, если

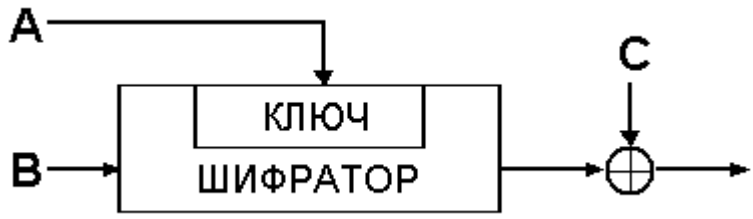
   B = 1234 5678 ABCD 2345 34564 5678 ABCD 2468
   C = 2234 5678 ABCD 2345 34564 5678 ABCD 2468
   D = 3234 5678 ABCD 2345 34564 5678 ABCD 2468

§ 5. ХЕШ-ФУНКЦИИ НА ОСНОВЕ БЛОКОВЫХ ШИФРОВ (КЛЮЧЕВЫЕ ХЕШ-ФУНКЦИИ)

Ключевые хэш-функции строят с помощью блокового $n-$битового шифра E_k (далее запись $E_k(X)$ означает, что шифр работает на ключе k). Сначала сообщение M дополняют до нужной длины и разбивают на блоки $x_1, x_2, ..., x_N$, длина которых совпадает с размером блока или ключа блокового шифра (это зависит от вида хэш-функции). При построении ключевой хеш-функции используют все туже схему Меркеля – Дамгарда:

$$H_0 = IV;$$
$$H_i = f(x_i, H_{i-1}),\ i = 1,...,t;$$
$$H(M) = H_t,$$

где IV – начальное значение свертки (фиксированный вектор), f – функция сжатия. Обобщенная схема формирования хеш-функции с помощью блокового шифра изображена на рисунке:



В роли A, B и C могут выступать блоки входного текста, результаты i-ой итерации свертки, а безопасность хеш-функции базируется на безопасности используемого шифра. Определение функции сжатия, участвующей в итерациях, зависит от используемой схемы. Наиболее безопасные хеш-функции такие:

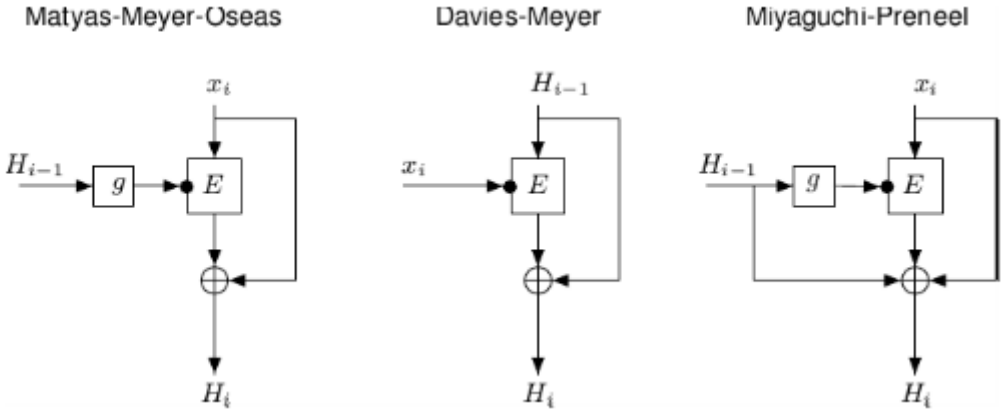
1) $f(x_i, H_{i-1}) = E_{g(H_{i-1})}(x_i) \oplus x_i$ – функция Матиаса – Мейєра – Осиса (роль ключа в шифре играют элементы свертки, полученной на предыдущей итерации);

2) $f(x_i, H_{i-1}) = E_{x_i}(H_{i-1}) \oplus H_{i-1}$ – функция Давиэса-Мейєра (роль ключа играют элементы сообщения);

3) $f(x_i, H_{i-1}) = E_{g(H_{i-1})}(x_i) \oplus x_i \oplus H_{i-1}$ – функция Миягучи-Пренила.

Здесь g некоторая функция, которая переводит вектор длины m в вектор длины n.

Основной недостаток хеш-функций, спроектированных на основе блоковых шифров, – низкая скорость работы (по сравнению с бесключевыми функциями).



§ 6. ОСНОВНЫЕ МЕТОДЫ КРИПТОАНАЛИЗА ОДНОСТОРОННИХ ХЭШ-ФУНКЦИЙ

1. **Лобовая атака**. *Имея свертку* $H(m_1)$ *сообщения* m_1, *криптоаналитик должен методом подбора найти сообщение* m_2 ($m_2 \neq m_1$), *для которого* $H(m_1) = H(m_2)$ *(тогда он может заявлять, что предъявленная свертка соответствует сообщению* m_2, *а не*

m_1). Если хэш-функция на выходе дает n-битовую строку, то сложность этого метода $O(2^n)$.

2. **Атака на основе «парадокса дней рождений»***. Для хэш-функций универсальным методом поиска коллизий является метод, основанный на известной статистической задаче – «парадоксе дня рождения».*

Парадокс дней рождения — это кажущееся парадоксальным утверждение, что вероятность совпадения дней рождения (даты) хотя бы у двух членов группы из 23 и более человек, превышает 0,5. Для 60 и более человек вероятность такого совпадения превышает 0,99, хотя 1,0 она достигает, только когда в группе не менее 367 человек. Такое утверждение может показаться неочевидным, так как вероятность совпадения дней рождения двух человек в любой день года (1/365 = 0,0027), помноженная на число человек в группе из 23, даёт лишь 23/365 = 0,063. Это рассуждение неверно, так как число возможных пар (253) значительно превышает число человек в группе. Логического противоречия в этом нет, а парадокс заключается лишь в различиях между интуитивным восприятием и математическим расчётом.

Рассчитаем сначала вероятность $\overline{P(n)}$ того, что в группе из n человек дни рождения всех людей будут различными. Если n > 365, то $\overline{P(n)} = 0$. При n ≤ 365 выберем наугад одного человека из группы и запомним его день рождения. Затем выберем наугад второго человека, при этом вероятность того, что у него день рождения не совпадёт с днем рождения первого человека, равна 1 — 1/365. Затем возьмём третьего человека, при этом вероятность того, что его день рождения не совпадёт с днями рождения первых двух, равна 1 — 2/365. Рассуждая по аналогии, мы дойдём до последнего человека, для которого вероятность несовпадения его дня рождения со всеми предыдущими будет равна 1 — (*n* — *1*)/365. Перемножая все эти вероятности, получаем вероятность того, что все дни рождения в группе будут различными:

$$\overline{P(n)} = 1 \cdot \left(1 - \frac{1}{365}\right) \cdot \left(1 - \frac{2}{365}\right) \cdot \ldots \cdot \left(1 - \frac{n-1}{365}\right) =$$

$$= \frac{365 \cdot 364 \cdot \ldots \cdot (365 - n + 1)}{365^n} = \frac{365!}{365^n (365 - n)!}.$$

Тогда вероятность того, что хотя бы у двух человек из n дни рождения совпадут, равна

$$P(n) = 1 - \overline{P(n)}.$$

Интересно, что при $n > 23$ эта вероятность $P(n) > 0,5$.

Используя разложение экспоненты в ряд Телора $e^x = 1 + x + \frac{x^2}{2!} + \ldots$,

полученное выражение можно аппроксимировать

$$\overline{P(n)} \approx 1 \cdot e^{-\frac{1}{365}} \cdot e^{-\frac{2}{365}} \cdot \ldots \cdot e^{-\frac{n-1}{365}} = e^{-\frac{1+2+\ldots+(n-1)}{365}} = e^{-(n(n-1))/2 \cdot 365}.$$

$$P(n) = 1 - \overline{P(n)} \approx 1 - e^{-(n(n-1))/2 \cdot 365}$$

и при более грубой оценке $P(n) \approx 1 - e^{-n^2/2 \cdot 365}$.

Сведем задачу криптоанализа хэш-функций к задаче поиска коллизии: сколько сообщений надо просмотреть, чтобы найти сообщения с двумя одинаковыми хэшами? Вероятность встретить одинаковые хэши для сообщений из двух разных наборов, содержащих n_1 и n_2 текстов, равна $P \approx 1 - e^{-\frac{n_1 n_2}{2^l}}$. Если $n_1 = n_2 = 2^{l/2}$, то вероятность успеха атаки $P \approx 1 - e^{-1} \approx 0,63$, а сложность проведения атаки $2^{\frac{l}{2}+1}$ операций. Чтобы найти коллизию, надо сгенерировать два псевдослучайных множества сообщений (в каждом множестве $2^{n/2}$ сообщений) и найти для них хэши. Тогда согласно парадоксу дня рождения, вероятность того, что среди них найдется пара сообщений с одинаковыми хэшами, больше 0,5. Атака требует большого объема памяти для хранения текстов и эффективных методов сортировки.

Примеры.

1. Противник перехватил хєш $H = H(M_1)$. Длина хэша n битов. Он хочет найти любое сообщение M_2, для котрого $H(M_1) = H(M_2)$, для чего генерирует k сообщений и вычисляет их хэши. Какова вероятность успеха?

   Ответ: вероятность успеха $P = 1 - e^{-k/n}$.

2. Противник перехватил определенное число хєшей разных сообщений. Длина хэша n битов. Сколько новых сообщений и их хэшей надо сгенерировать, чтобы найти коллизию для 50 % перехваченных хешей?

   Ответ: $k \approx 0,63 \cdot 2^n$.

3. Противник перехватил хєш $H = H(M_1)$ и сообщение M_1. Он хочет найти другое сообщение M_2, для котрого

$H(M_1) = H(M_2)$, для чего генерирует $k-1$ сообщений и вычисляет их хэши. Какова вероятность успеха?

Ответ: Вероятность успеха $P = 1 - e^{1-(k-1)/n}$.

4. Противник хочет создать два любых разных сообщения M_1 и M_2, для которых $H(M_1) = H(M_2)$, для чего генерирует k сообщений и вычисляет их хэши. Какова вероятность успеха?

Ответ: Вероятность успеха $P = 1 - e^{1-(k-1)/2n}$.

5. Хэш-функция дает хэш длиной 64 бита. Сколько хэшей надо сгенерировать, чтобы найти коллизию двух любых сообщений?

Ответ: $k \approx 1{,}18 \cdot 2^{n/2} \approx 1{,}18 \cdot 2^{32}$. Если проверять 220 сообщений за секунду (почти 10^6 опер/сек.), то потребуется $\approx 1{,}18 \cdot 2^{12}$ секунд $<$ 2 часов. Хэш сообщения в 64 бита небезопасен относительно атаки коллизии.

Сценарий проведения атаки хеш-функции, на основе «парадокса дня рождения». Пусть мошенник **A** хочет подвергнуть атаке банкира **B** и подписать у него контракт, невыгодный для **B**.

1). **A** готовит две версии контракта – «хорошую» M_1 и «плохую» M_2, делает несколько малозначительных изменений к каждому документу, манипулируя с точками, запятыми, пробелами, предлогами и др.

2). **A** считает для всех версий контракта хэши, сравнивает наборы хэшей, подыскивая одинаковые пары. Если длина хэша 64 бита, то обычно хватает 2^{32} пар. **A** выбирает пару с одинаковым хэшем.

3). **A** выбирает пару с одинаковым хэшем и дает на подпись банкиру **B** плохую версию M_2. Тогда **A** в суде может доказывать, что **B** подписал невыгодный контракт сознательно.

При работе с ключевыми хэш-функциями небезопасно дописывать ключ в начало или конец исходного сообщения.

- Пусть ключ k добавлен в начале сообщения, а сжимающая функция построена по схеме Меркеля – Дамгарда. Тогда по известным сообщению M и его свертке $H = H(k \| M)$ можно найти значения свертки для всех сообщений вида $M \| M'$, где после M дописано любое сообщение M'. Действительно, в силу итерационного характера хеш-функции для нахождения $H' = H(k \| M \| M')$ не нужно знать ключ. Достаточно использовать у вычисленное «промежуточное» значение H.

▪ Пусть ключ k добавлен в конец сообщения $H = H(M \| k)$. Знание коллизии для хэш-функции, т.е. пары M_1, M_2, $M_1 \neq M_2$, но $H(M_1) = H(M_2)$, позволяет вычислить хэши $H(M_1, k) = H(M_2, k)$ для любого ключа k. $\Rightarrow$ трудоемкость изменения сообщения M_1 уже оценивается не величиной $O(2^n)$. Атакуя хэш-функцию на основе «парадокса дней рождения» можно снизить трудоемкость до $O(2^{n/2})$.

Чтобы уйти от этих опасностей, рекомендуют ключ приписывать несколько раз:

$$H = H(k \| y \| M \| k) \quad \text{или} \quad H = H(k \| y_1 \| H(k \| y_2 \| M)),$$

где y, y_1, y_2 – дополнение ключа до размера, кратного длине блока. Так построенные бесключевые функции устойчивы к атакам коллизии, но их недостаток – большая длина свертки.