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.

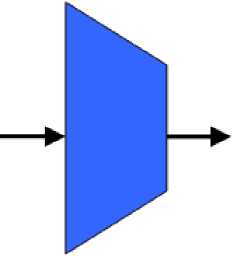
Различают несколько способов идентификации: контрольные суммы, контроль CRC, хэширование и цифровая подпись - таковы базовые средства аутентификации при цифровой передаче данных.

***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*.

**1A3FD4128A198FB3CA345922**

Это вход хэш-функции. Вход - это строки бит произвольной длины, ко­торую хэш-функция превращает в строку бит фиксированной длины

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

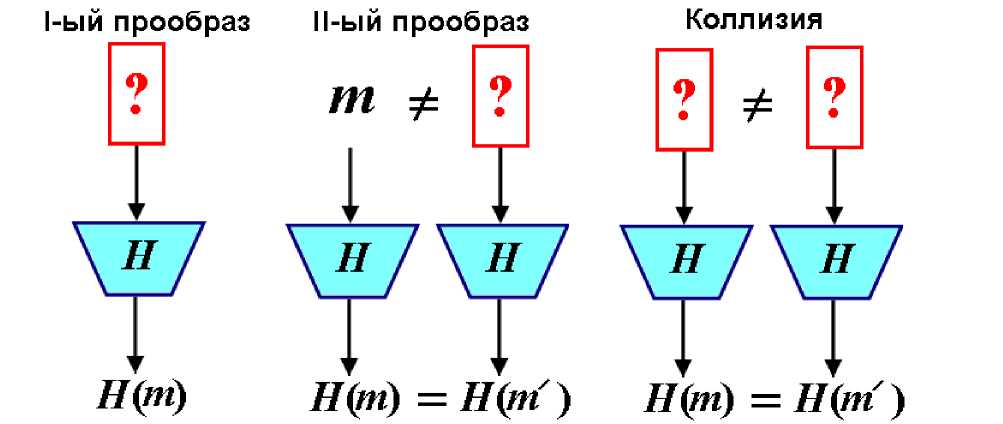
В литературе можно широко распространены и другие названия, а именно

***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 Ф 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* Ф *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 ^ коллизии для хэш- функций обязательно существуют. Например, пусть длина хэш- прообраза 6 битов, длина свертки 4 бита. Тогда число различных



сверток - 24 *=* 16, а число хэш-прообразов - 26 *= 64,* т.е. в 4 раза

больше ^ 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 **А** подает сообщение на вход функции хэширования и находит его свертку (хэш).
* Свертка сообщения добавляется к сообщению.
* Отправитель **А** отправляет получателю сообщение+свертку .
* Получатель пропускает сообщение через функцию хэширования.
* Получатель **В** 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 **В**. 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 (МАС - 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.

1. 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.

1. Хэш-функция *H(m) = m*(mod*n), H(m1)* и *H*(*m*2) - свертки сообщений *m1* и *m*2. Можно ли выбрать в роли значения свертки сообщения *m3* = *m1* + *m*2 значение (*H (m1)* + *H* (*m*2))(mod *n)* ?

*Ответ:* Нет, нарушается свойство перемешивания.

1. 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 *у = f (x1,x2),* где *x1,x2 -* битовые строки длины *m, у -* 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,..., *Mt.* Свертки, получаемые в ходе итераций, обозначим *H1,H2,...,Ht*, результирующая свертка всего сообщения *H (M*). Последовательная процедура вычисления свертки:

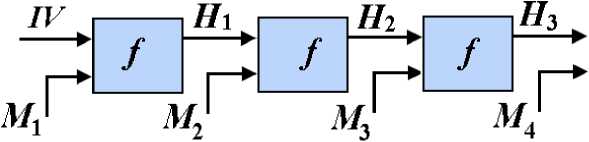
*H*0 = *IV*;

*Hi = f(Mi,Hi-i)*. *і =* і,...,*t*;

*H (M) = Ht*,

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.



***т*** бит ***т*** бит

*\*Н,=Н(М)*

§ 4. ДВА ТИПА КРИПТОГРАФИЧЕСКИХ ХЭШ-ФУНКЦИЙ

Типы криптографических хэш-функций

***Ключевые хеш-функции***или

***коды аутентификации  
сообщений***

*(****Message Аиїйепїісаїіоп***

***Code - MAC****)*

Используются в системах с доверяющими друг другу пользователями

***Бесключевые хеш-функции*** или

***коды обнаружения ошибок****(****Modification Code - MDC***или

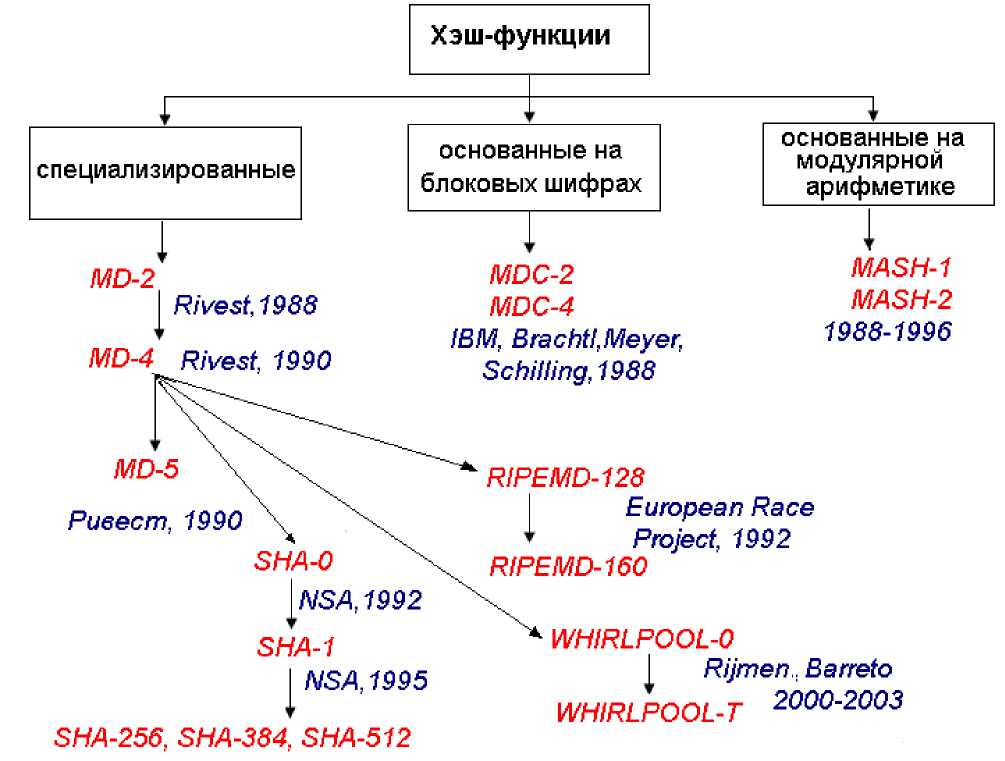
***Message Integrity Code -MIC****)* Используются в системах как с доверяющими друг другу, так и с недоверяющими пользователями

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. |
| NAVAL | 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:

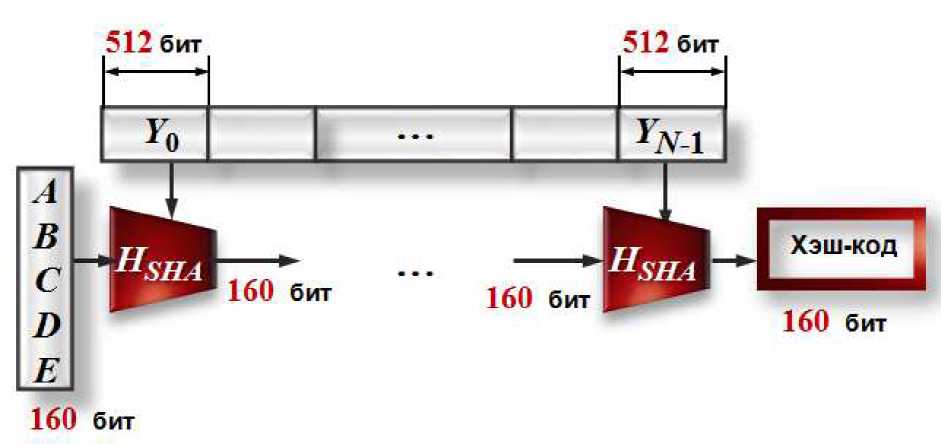
Длина хэш-кода - 160 бит

Длина обрабатываемых блоков - 512 бит

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

Максимальная длина хэшируемых данных - 264 -1.

1. Логика выполнения SHA-1



Алгоритм получает на входе сообщение максимальной длины 264 — 1 бит и создает в качестве выхода дайджест сообщения длиной 160 бит.

Алгоритм состоит из следующих шагов:

Логика выполнения SHA-1

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

Исходное *m*-битовое сообщение *(a1,a2,...,am*)

расширяется до

длины, кратной 512, по принципу:

*(a1*,*a2,---,am* Д^0^0,*/*1, */*2

*^*64) ,

где *(/1,/2,...,/64) -* двоичная запись числа *m*, *5*- наименьшее число, при котором (*m* + *s* + 64)M12.

**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 = 0(mod512) ^ *s* = *-m -* 64(mod512)

*s* = -2590 - 64(mod512) = -2654(mod512) = 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(mod512) = 0, тогда *m* = -64 = 448(mod512). Максимальная длина заполнения - 511 битов, когда *-m -* 64(mod512) = 511(mod512) и *m =* 449(mod512)

Итак, результатом первого шага является сообщение, длина которого кратна 512 битам. Далее расширенное сообщение делится на 512-битные блоки *Y0,Y1,...,Yn\_1 ,* поэтому общая длина расширенного сообщения есть *N ■* 512 бит. Этот результат кратен шестнадцати 32­битным словам.

1. 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 2301;

*B = EF CD AB* 89;

*C* = 98 *BA DC FE*;

*D* = 10 32 54 76;

*E = C*3 *D*2 *E*1 *F*0 .

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

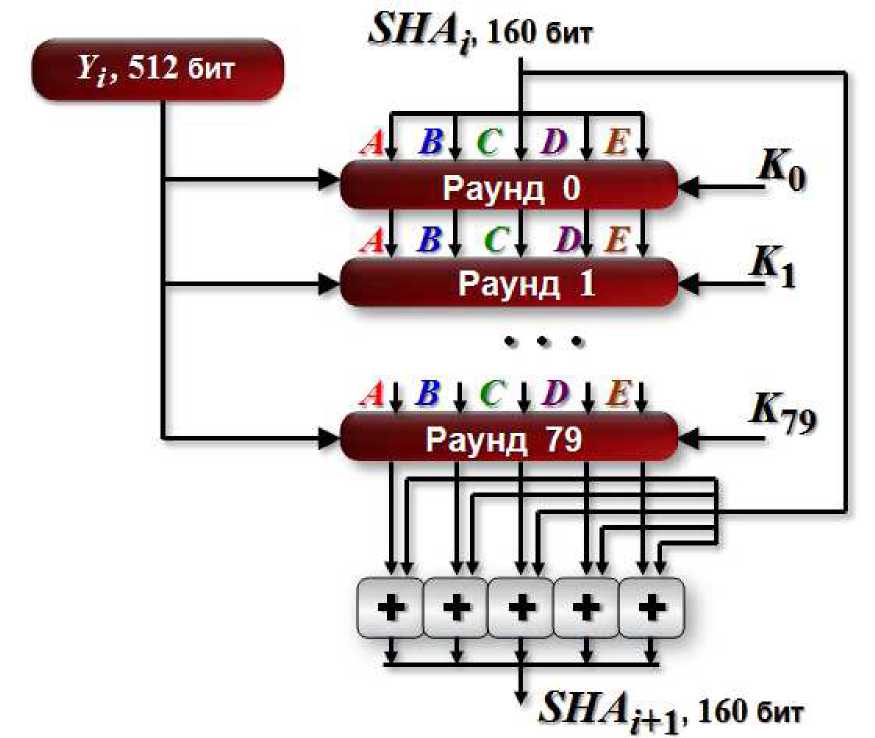
*SHA0* = *A*|| *B*||*C* ||*D* ||*E*.

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

Основой алгоритма является модуль, состоящий из 80 циклических обработок, обозначенный как *HSHA*. Все 80 циклических обработок имеют одинаковую структуру.

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

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



Обработка очередного 512-битного блока

5*A*827999, 0 < *t <* 19;

6 *ED9 EBA1, 8F1BBCDC*,

20 < *t <* 39,

40 < *t* < 59;

*CA*62*C*1*D*6, 60 < *t* < 79.

Для получения *SHAi+1* выход 80-го цикла складывается со значением *SHAj.* Сложение по модулю 232 выполняется независимо для каждого из пяти слов в буфере с каждым из соответствующих слов в *SHAj.*

1. Выход (шаг 4)

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

1. 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, Kt ,Wt*) = *S* 5( *A)* + *ft* (*B, C, D)* + *E* + *Kt* + *Wt*;

*E = D*; *D = C*; *C = S30(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;

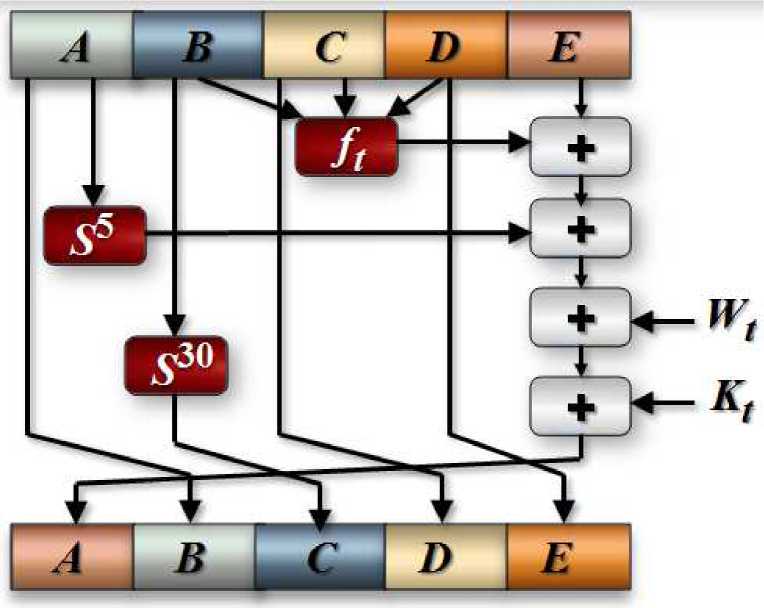
32

символ + означает сложение по модулю 2 ;

*ft* - элементарная логическая функция (описана ниже);

*Kt* - пошаговые константы, равные

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



Логика работы одного раунда

Замечание: значения пошаговых констант *Kt* в алгоритме были выбраны по принципу ([*х*] - целая часть числа)

1. *A*827999 = [230 -V2],
2. *ED9 EBA1* = [230 -УЗ]*,*

*8F1BBCDC* = [230 ^/5]*,*

*CA62C1D6* = [230 -710]*.*

6. Логические функции *ft,* 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):

(*B* A *C*) v (*B* A *D*),

*ft* (*B*, *C*, *D*) = ^ *B* Ф *C* Ф *D*,

0 < *t* < 19;

20 < *t <* 39, 60 < *t <* 79;

(*B* a *C*) v (*B* • *D*) v (*C* a *D*), 40 < *t <* 59.

(*B* получено из *B* инвертированием бит).

**Пример.** Крайние левые шестнадцатеричные цифры переменных *B*,*C*,*D* равны *07 h*, 0*Ah* и 0*Eh* соответственно. Какие цифры будут крайними левыми в значении функции *f43(B,C,D)*, используемой в алгоритме SHA?

Р е ш е н и е. 07*h* = 01112, *0Ah =* 10102, *0Eh =* 11102.

(*B a C*) v (*B a D) v (C a D) =* (0 a 1) v (0 a 1) v (1 a 1) = 0 v 0 v 1 = 1.

(1 a 0) v (1 a 1) v (0 a 1) = 0 v 1 v 0 = 1;

(1A1) v (1A1) v (1A1) = 1 v 1 v 1 = 1;

(1A 0) v (1A 0) v (0 A 0) = 0 v 0 v 0 = 0.

Результат 11102 = 0 *Eh*.

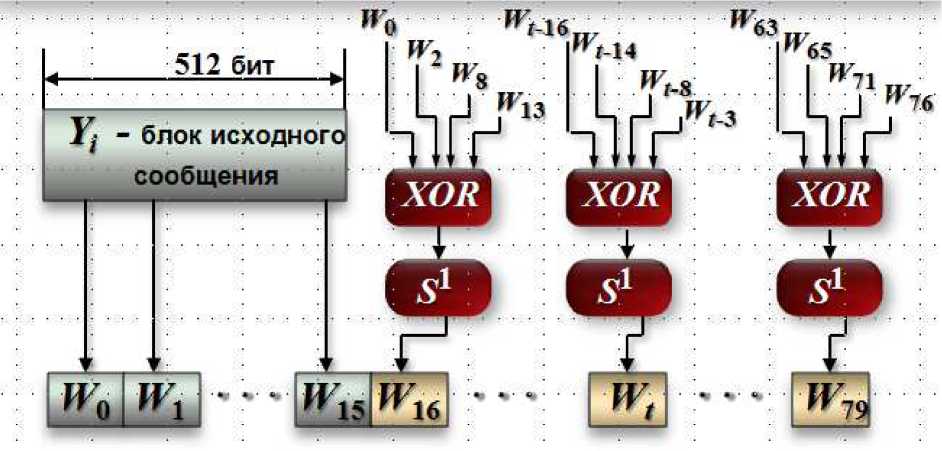
1. Формирование входных значений  
   раунда из очередного блока

Перед началом первого раунда блок расширяют до 80 слов по 32 бита. Для этого вначале сам входной блок данных делят на 16 слов *Y0,Y1,..., Y*15 по 32 бита и далее к ним присоединяют еще 64 слова по 32 бита. В результате образуются слова: *W0,W1,...,W79.* Принцип образования новых слов:

*Wt = Yt*, при *t =* 0,1,...,15;

*Wt = Sl(Wt\_3* Ф *Wt\_8* ф *Wt*\_14 Ф *Wt\_16)* при *t =* 16,...,79,

где *S*1 - операция циклического сдвига аргумента на один разряд влево.



Алгоритм *SHA-1* можно суммировать следующим образом:

*SHAi+1 =* ^ *32(SHAi*, *ABCDEi*);

*SHA* = *SHAn \_1,* где

*IV* - начальное значение буфера *ABCDE*;

*ABCDEi* - результат обработки */*-того блока сообщения;

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

^ 32 - сумма по модулю 232, 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:

о 5120 битов

о 5121 бит

о 6143 бита

1. Найдите результат функции *f47(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

1. 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 *Ek* (далее запись *Ek* (*X)* означает, что шифр работает на ключе *k*). Сначала сообщение *M* дополняют до нужной длины и разбивают на блоки *x1,x2,...,xN*, 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*;

*Ht = f(xt , Ht-})*, *i = 1,..., /*;

*H (M)* = *Ht*,

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

ключ

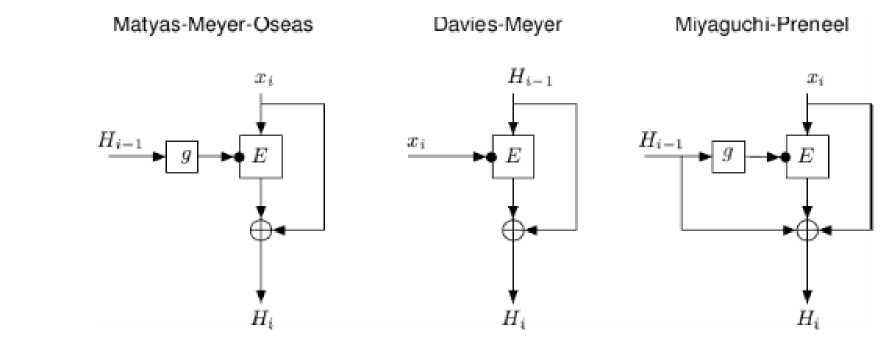
В роли *A, B* и *C* могут выступать блоки входного текста, результаты *i* - ой итерации свертки, а безопасность хеш-функции базируется на безопасности используемого шифра. Определение функции сжатия, участвующей в итерациях, зависит от используемой схемы. Наиболее безопасные хеш-функции такие:

ШИФРАТОР

1. *f* (*xi, Hi* \_1) *= Eg (H, 1)(xi*) © *xi* - функция Матиаса - Мейєра - Осиса (роль ключа в шифре играют элементы свертки, полученной на предыдущей итерации);
2. *f (xi, Hi*\_1) *= Ex (Hi*\_1) © *Hi*\_1 - функция Давиэса-Мейєра (роль ключа играют элементы сообщения);
3. *f (xi,Hi*\_1) = *Eg(Hi\_x)(xi*) ©*xi* ©*Hi*\_1 - функция Миягучи-Пренила.

Здесь *g* некоторая функция, которая переводит вектор длины *т* 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(т1) сообщения т1, криптоаналитик должен методом подбора найти сообщение т2 (т*2 *Ф т1), для которого H(т1) = H*(*т*2) *(тогда он может заявлять, что предъявленная свертка соответствует сообщению т2, а неm1). Если хэш-функция на выходе дает n -битовую строку, то сложность этого метода O(2n*)*.*
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 *P(n)* того, что в группе из *n* человек дни рождения всех людей будут различными. Если *п* > 365, то *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 — (*п — 1*)/365. Перемножая все эти вероятности, получаем вероятность того, что все дни рождения в группе будут различными:

*— (*

*P(n)* = 1 • 1 - к

1 Ї (

•1

365) к

л

365 )

(1 - — 1  
к 365 )

365 • 364 •... • (365 - *n* +1)\_ 365!

365*n* 365*n* (365 - *n*)!

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

*P*(*n*) = 1 - *P*(*n*).

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

\_ \_ -г *x 1 . x2 t*

Используя разложение экспоненты в ряд Телора *e =* 1 + *x* + — +...,

полученное выражение можно аппроксимировать

\_ 1 \_ 2 *n*-1 \_ 1+2+...+(*n*-1)

*Pn)* = 1 • *Є* 365 . *e* 365 ..... *e* 365 = *e* 365 = *e~* (*n*(*n*-1)V2-365 **.**

*P(n) =* 1 - *P*(*n*) = 1 - *e~* -1))/M65

*n* 2/2.365

и при более грубой оценке *P*(*n*) ~ 1 - *e*

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*n*2

*n*1 и *n*2 текстов, равна *P \** 1 -*e 2* . Если *n*1 = *n*2 = *21* /2, то вероятность успеха атаки *P* ~ 1 - *e*-1 ~ 0,63, а сложность проведения атаки *l*+1

22 операций. Чтобы найти коллизию, надо сгенерировать два псевдослучайных множества сообщений (в каждом множестве 2*п*/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(М1).* Длина хэша *n* битов. Он хочет найти любое сообщение *М*2, для котрого *H*(*М*1) = *H*(*М*2), для чего генерирует *к* сообщений и вычисляет их хэши. Какова вероятность успеха?

*Ответ:* вероятность успеха *P =* 1 - *e~k*/*n*.

1. 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: к* ~ 0,63 • 2*п*.

1. Противник перехватил хєш *H = H* (*М*1) и сообщение *М*1. Он хочет найти другое сообщение *М*2 , для котрого *H (М1) = H* (*М*2), для чего генерирует *k -*1 сообщений и вычисляет их хэши. Какова вероятность успеха?

*Ответ:* Вероятность успеха *P =* 1 - *e*1-(*k*-1)/*n.*

1. Противник хочет создать два любых разных сообщения *М1* и *М2,* для которых *H(M1) = H(М*2), для чего генерирует *k* сообщений и вычисляет их хэши. Какова вероятность успеха?

*Ответ:* Вероятность успеха *P =* 1 -*e*1-(*k*-1)/2*n*.

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

*Ответ: к* ~ 1,18 • *2п*/2 ~ 1,18 • 232. Если проверять 220 сообщений за секунду (почти 106 опер/сек.), то потребуется ~ 1,18 • 212 секунд < 2 часов. Хэш сообщения в 64 бита небезопасен относительно атаки коллизии.

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

1. . **А** готовит две версии контракта - «хорошую» *М*1 и «плохую» *М*2, делает несколько малозначительных изменений к каждому документу, манипулируя с точками, запятыми, пробелами, предлогами и др.
2. . **А** считает для всех версий контракта хэши, сравнивает наборы хэшей, подыскивая одинаковые пары. Если длина хэша 64 бита, то обычно хватает 232 пар. **А** выбирает пару с одинаковым хэшем.
3. . **А** выбирает пару с одинаковым хэшем и дает на подпись банкиру **В** плохую версию *М*2. Тогда **А** в суде может доказывать, что **В** подписал невыгодный контракт сознательно.

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

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

**■** Пусть ключ *к* добавлен в конец сообщения *H = H*(*M* || *к*). Знание коллизии для хэш-функции, т.е. пары *M1,M2, M1 ф M2,* но *H(M1) = H(M2),* позволяет вычислить хэши *H(M1,к) = H(M2,к)* для любого ключа *к.* ^ трудоемкость изменения сообщения *M1* уже оценивается не величиной *O(2n*). Атакуя хэш-функцию на основе «парадокса дней рождения» можно снизить трудоемкость до *O(2n*/2).

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

*H = H&M* |*M*| |*к*) или *H = Щк\* |*у*1 ^(*к*^ ||*M*)), где *у,у1,у2 -* дополнение ключа до размера, кратного длине блока. Так построенные бесключевые функции устойчивы к атакам коллизии, но их недостаток - большая длина свертки.