СПбГЭТУ «ЛЭТИ»
ПЕРВЫЙ ЭЛЕКТРОТЕХНИЧЕСКИЙ

А.Б. Левина

Криптография и криптографические протоколы

Протоколы аутентификации

СПбГЭТУ «ЛЭТИ», 2022 г.

1. ВВЕДЕНИЕ

Мы продолжаем знакомство с асимметричным шифрованием и на этой лекции мы познакомимся с протоколом Диффи-Хеллмана, протоколом, с которого началась история асимметричного шифрования. Также мы познакомимся с электронно - цифровой подписью (цифровой подписью) и рассмотрим некоторые алгоритмы постановки подписи. На данной лекции мы также изучим основные принципы создания хэш-функций и рассмотрим уязвимости криптографических хэш-функций.

Сегодня на лекции будет рассмотрено:

- Протокол распределений ключей Диффи-Хеллмана;
- Цифровая подпись;
- Алгоритмы цифровой подписи - RSA и DSA;
- Подпись Шнорра;
- Что такое хэш-функции.

2. РАСПРЕДЕЛЕНИЕ КЛЮЧЕЙ - ПРОТОКОЛ ДИФФИ-ХЕЛЛМАНА

Прежде чем переходить к знакомству с протоколом Диффи-Хеллмана введем определение:

Определение 2.0.1 *Криптосистема обладает* ***прогрессивной секретностью****, если компрометация долгосрочного секретного ключа, в какой-то момент времени, не приводит к вскрытию тайны прошлой переписки.*

Как мы видим, асимметричное шифрование прогрессивной секретностью не обладает. Давайте познакомимся с протоколом Диффи-Хеллмана, с которого началось развитие асимметричного шифрования.

Определение 2.0.2 *Протокол Диффи—Хеллмана (англ Diffie-Hellman, DH) — криптографический протокол, позволяющий двум и более сторонам получить*

общий секретный ключ используя незащищенный от прослушивания канал связи. Полученный ключ используется для шифрования данных с помощью алгоритмов симметричного шифрования.

Протокол был предложен в 1976 г. Диффи и Хеллманом в их работе «New Directions in Cryptography».

Давайте вспомним задачу Диффи и Хеллмана и задачу дискретного логарифмирования, о которых мы говорили на прошлой лекции.

ПДЛ - проблема дискретного логарифмирования:

Дано: A, B ∈ G, G конечная Абелева группа (G,).

Найти: x: $B = A^x$.

В задаче Диффи-Хеллмана группа G = Z_P является кольцом вычетов по модулю P.

ЗДХ - задача Диффи-Хеллмана:

Дано: $B = A^x$, $C = A^y$;

Найти: $D = A^{x \cdot y}$.

Посмотрев ЗДХ мы понимаем, чтобы ее решить необходимо найти x и y, а это возможно только если уметь решать задачу дискретного логарифмирования.

Рассмотрим сам протокол.

2.1 Распределение ключей Диффи-Хеллмана.

В системе генерируются параметры домена - общие параметры, используемые всеми пользователями G, P.

Рассмотрим пошагово действие Алисы и Боба:

1. Алиса генерирует свое **a** и вычисляет $G^{\mathbf{a}}(mod\ P)$ и высылает полученные данные Бобу.

2. Боб генерирует свое **b** и вычисляет $G^{\mathbf{b}}(mod\ P)$ и высылает полученные данные Алисе.

3. После этих двух шагов у Алисы есть **a** и $G^{\mathbf{b}}(mod\ P)$, полученное от Боба, а у Боба есть **b** и $G^{\mathbf{a}}(mod\ P)$, полученное от Алисы.

4. Алиса возводит полученное от Боба $G^{\mathbf{b}}(mod\ P)$ в степень **a** и получает $G^{\mathbf{b}\cdot\mathbf{a}}(mod\ P)$.

5. Боб возводит полученное от Алисы $G^a(mod\ P)$ в степень **b** и получает $G^{b \cdot a}(mod\ P)$. После этого шага у Алисы и у Боба есть общий ключ $k = G^{b \cdot a}(mod\ P)$.

Схема протокола генерации ключа представлена на рисунке 1.

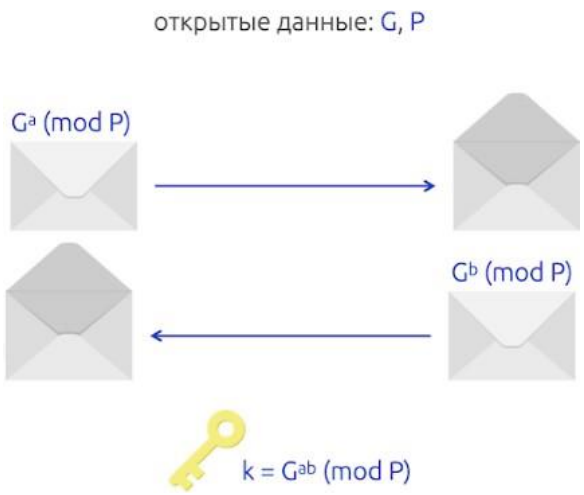

Рис. 1: Протокол Диффи-Хеллмана

Данный протокол не только обеспечивает безопасное создание ключа, но и позволяет каждому пользователю участвовать в генерации ключа, тем самым повышая уровень доверия к общему ключу.

Очень хорошо работу протокола можно проследить на примере с красками, представленном на рисунке 2.

Рис. 2: Протокол Диффи-Хеллмана, пример с красками

Рассмотрим работу протокола на примере с большими числами, чтобы максимально приблизиться к реальным условиям.

Пример 2.0.1 *P=2 147 483 659, G=2.*

Алиса генерирует a = 12345, *Боб генерирует* b = 654323.

Алиса вычисляет $G^a(mod\ P)$, *получает* 428647416 *и пересылает это Бобу.*

Боб вычисляет $G^b(mod\ P)$, *получает* 450904856 *и пересылает это Алисе.*

Алиса возводит 450904856 *в свою степень* a = 12345 *и получает* 1333327162.

Боб возводит 428647416 *в свою степень* b = 654323 *и тоже получает* 1333327162.

Как видно из примера и у Алисы и у Боба есть общий ключ.

Мы специально рассмотрели пример с большими числами, чтобы было понятно, с данными какого объема работают реальные системы.

Мы познакомились с протоколом Диффи-Хеллмана, основанного на задаче Диффи-Хэллмана, а так как задача ДХ сводится к задаче дискретного логарифмирования, которая за полинамиальное время не решается, то мы получили абсолютно безопасный протокол, но так ли это?

На самом деле нет, протокол ДХ абсолютно беззащитен перед атакой "человек посередине"/"man in the middle (MITM)".

Рассмотрим эту атаку.

На первом шаге Алиса генерирует свое **a**, вычисляет $G^a(mod\ P)$ и высылает полученные данные Бобу, но какая у нее может быть уверенность, что эти данные пришли именно к Бобу, а не к Еве?

Рассмотрим это на рисунке 3.

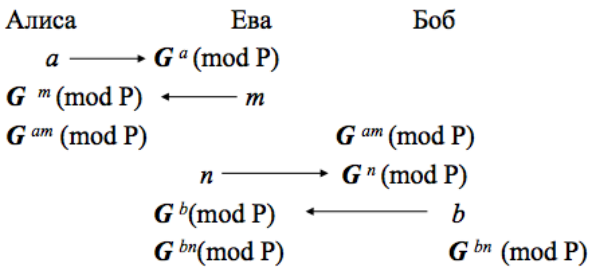

Рис. 3: Атака "человек посередине"

1. Алиса договаривается с Евой, думая, что она общается с Бобом;
2. Боб также ведет переговоры с Евой, считая, что это Алиса;
3. Ева читает все сообщения, так как они проходят через нее как через коммутатор. При этом она может их менять, или не менять по своему усмотрению. Если Ева не вносит изменений в сообщения, то обнаружить ее невозможно.

Для решения этой проблемы необходимо пересылаемые данные подписывать, это обеспечивается с помощью алгоритмов электронной цифровой подписи, или, как часто сокращенно называют, алгоритмов электронной подписи. С ними мы сейчас и познакомимся.

3 ЦИФРОВАЯ ПОДПИСЬ

3.1 СХЕМА ЦИФРОВОЙ ПОДПИСИ

В криптографии есть два вида схем цифровой подписи. Познакомимся с ними:

1. Схема подписи **с приложением**;
2. Схема подписи **с восстановлением сообщения**.

Схема подписи с приложением применяется в тех случаях, когда нас интересует только факт подписания документа, само сообщение, в данном случае, не интересует. Важна только информация, кто является отправителем.

Сообщение подписывается с помощью закрытого ключа, а проверяется подпись с помощью открытого ключа, получается, что проверить подпись может любой.

СООБЩЕНИЕ + секретный ключ Алисы = ПОДПИСЬ
СООБЩЕНИЕ + ПОДПИСЬ + ОТКРЫТЫЙ КЛЮЧ АЛИСЫ = ДА/НЕТ

Рис. 4: Схема подписи с приложением

Схема подписи с восстановлением не только предоставляет информацию, кому принадлежит сообщение, но и восстанавливает сообщение, в случае если подпись является ликвидной.

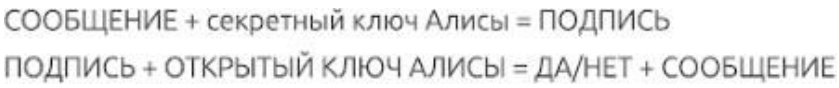

Рис. 5: Схема подписи с восстановлением

Цифровая подпись должна удовлетворять следующим критериям:

1. Подпись достоверна. Она убеждает получателя документа в том, что подписавший осознанно подписал документ.
2. Подпись неподдельна. Она доказывает, что именно подписавший, и никто иной, подписал документ.
3. Подпись не может быть использована повторно. Она является частью документа, ее невозможно перенести на другой документ.
4. Подписанный документ нельзя изменить.
5. От подписи нельзя отказаться. Подписавший не сможет утверждать, что он не подписывал документ, так как документ подписывается закрытым ключом, а пара закрытый ключ - открытый ключ уникальна.

Схематически это выглядит следующим образом:

- Алиса подписывает сообщение с помощью своего закрытого ключа, используя любой из алгоритмов постановки подписи. $S = s(M)$ -цифровая подпись.
- Боб осуществляет проверку подписи V (S), используя открытый ключ Алисы и получает на выходе сообщение M и бит v - результат проверки подписи.

Подпись гарантирует:

1. Целостность сообщения.

2. Оригинальность сообщения.

3. Отсутствие ренегатства=невозможность отказаться.

Когда мы сейчас знакомились с подписями, мы всегда говорили, что подпись ставится на сообщение. На самом деле это не так, и сейчас мы рассмотрим почему. Подпись ставится с помощью алгоритмов асимметричного шифрования, а на прошлой лекции мы обсуждали, что это очень время затратные преобразования. Если мы будем подписывать сообщение с помощью алгоритмов асимметричного шифрования, а потом еще шифровать подпись, то системы будут работать очень медленно и будут абсолютно не применимы в реальных условиях. Поэтому подпись ставится на хэш от сообщения.

Определение 3.1.1 *Хэш-функцией (хэш-кодом) называется преобразование, переводящие битовую строку любой длины в битовую строку фиксированной длины.*

Что такое хэш - функциях и хэш от сообщения более подробно мы узнаем на следующей лекции, а сейчас просто запомним, что мы ставим подпись не на само сообщение, а на его сжатый прообраз.

Рассмотрим это на схеме:

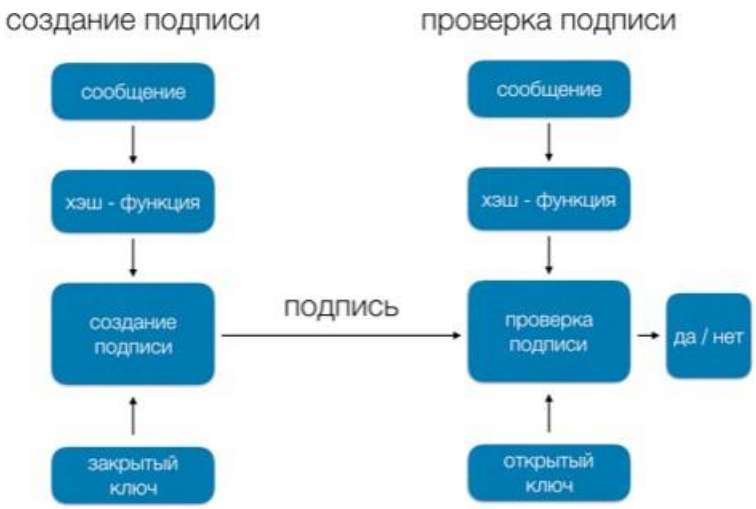

Рис. 6: Алгоритм постановки подписи

Мы познакомились с основными принципами работы электронной подписи. Давайте

теперь рассмотрим конкретные алгоритмы. Первый алгоритм, с которым мы познакомимся – уже известный нам алгоритм RSA, но сейчас мы рассмотрим его не как алгоритм шифрования, а как алгоритм постановки подписи.

3.2 RSA ДЛЯ ПОДПИСИ

Для начала давайте вспомним алгоритм RSA.

Алиса генерирует два больших простых числа p, q и вычисляет $N = p\ q$. Далее Алиса выбирает простое число E: НОД (E, (p-1)(q-1)) = 1. Доступная пара/открытый ключ – (N, E).

После этого Алиса находит d - расшифровывающую экспоненту, такую, что $E\ d = 1(mod\ (p-1)(q-1))$, d находится с помощью расширенного алгоритма Евклида.

Секретным ключом является тройка (d, p, q).

Процесс шифрования идет следующим образом:

1. Шифротекст C открытого текста $m < N$ вычисляется по формуле $C = m^E(mod\ N)$.

2. Для дешифрования шифротекст C возводится в степень d и получается обратно открытый текст $m = C^d(mod\ N)$.

Рассмотрим как идет процесс постановки подписи с помощью алгоритма RSA.

1. Для постановки подписи Алиса вычисляет хэш от сообщения $h(m)$, подписывает его используя свой **закрытый ключ** d и ставит свою подпись:
$S = h(m)^d(mod\ N)$.

2. Для проверки подписи Боб вычисляет $m' = S^E(mod\ N)$, и если $m' = h(m)$ то подпись корректна.

Это представлено на рисунке 7.

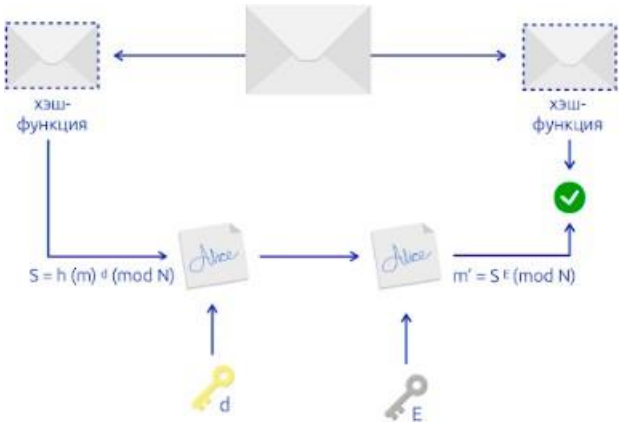

Рис. 7: Алгоритм RSA для постановки подписи

Но RSA для подписи не очень прижилось и на практике чаще используют другие алгоритмы. Давайте познакомимся с ними.

3.3 АЛГОРИТМ DSA

Определение 3.3.1 ***DSA*** *(Digital Signature Algorithm — алгоритм цифровой подписи) — криптографический алгоритм с использованием открытого ключа для создания электронной подписи.*

В отличие от RSA, алгоритм DSA используется только для подписывания документов, для шифрования он не используется. Алгоритм основан на вычислительной сложности взятия логарифмов в конечных полях. Алгоритм был предложен NIST в августе 1991 и является стандартом. Автор алгоритма David W. Kravitz.

DSA является частью DSS (Digital Signature Standard — стандарт цифровой подписи), впервые опубликованного 15 декабря 1998. Стандарт несколько раз обновлялся, последняя версия FIPS-186-4 была опубликована в июле 2013.

Разберем алгоритм по шагам:

1. (a) Фиксируется 160-битовое простое Q;

   (b) Выбирается простое P: $2^{512} < P < 2^{2048}$, такое что P-1 делится на Q;

   (c) Генерируется $L < P$;

(d) Вычисляется $G = L^{(P-1)/Q}(mod\ P)$ если $G = 1$ то идем на шаг 3.

(P, G, Q) - общие параметры домена

2. (a) Генерируется x: $0 < x < Q$;

(b) Открытый ключ Y: $Y = G^x(mod\ P)$.

3. Подпись сообщения

(a) Вычисляется $H = h(m)$;

(b) Выбирается эфемерный ключ $0 < k < Q$;

(c) Определяется $R = [G^k(mod\ P)](mod\ Q)$;

(d) Находится $S = (H + xR)/k(mod\ Q)$.

Подписью сообщения m служит (R, S) - 320 двоичных знаков

4. Проверка подписи

(a) Вычисляется $H = h(m)$;

(b) Определяется $A = H/S(mod\ Q)$, и $B = R/S(mod\ Q)$;

Находится $V = [(G^A)(Y^B)(mod\ P)](mod\ Q)$, Y - открытый ключ автора сообщения;

(d) Подпись корректна если $V = R$. Рассмотрим работу алгоритма на примере.

Пример 3.3.1

1. *(a) Пусть* $Q = 13$

(b) Выбирается простое $P = 53$

(c) $G = 16$.

(53, 16, 13) - *общие параметры домена*

2. *(a)* $x = 3$

(b) Открытый ключ Y: $Y = 16^3(mod\ 53) = 15$

3. *Подпись сообщения*

(a) Пусть $H = h(m) = 5$

(b) Эфемерный ключ $k = 2$

(c) Определяется $R = [16^2(mod\ 53)](mod\ 13) = 5$

(d) Находится $S = (5 + 3 \cdot 5)/2(mod\ 13) = 10$

Подписью сообщения будет (5, 10).

4. *Проверка подписи*

(a) $H = 5$

(b) $A = 5/10(mod\ 13) = 7$, *и* $B = 5/10(mod\ 13) = 7$

(c) $V = [(16^7)(15^7)(mod\ 53)](mod\ 13) = 5$

(d) Подпись корректна так как $V = 5 = R = 5$

Мы познакомились с алгоритмом DSA и сейчас рассмотри еще один алгоритм подписи - алгоритм Шнорра.

3.4 ПОДПИСЬ ШНОРРА

Схема Шнорра/подпись Шнорра (schnorr scheme)— одна из наиболее эффективных и теоретически обоснованных схем аутентификации, безопасность схемы основывается на трудности вычисления дискретных логарифмов.

Схема Клауса Шнорра является модификацией схем Эль-Гамаля (1985) и Фиата-Шамира (1986), но имеет меньший размер подписи.

Рассмотрим подпись Шнорра.

Первый шаг – создание ключевой пары секретный ключ - открытый ключ происходит также как и в подписи DSA:

(P, Q, G) - общие параметры домена;

$x : 0 < x < Q$ - секретный ключ; открытый ключ

$Y = G^x(modP)$.

Но процесс постановки подписи и проверки отличаются от алгоритма DSA.

1. Подпись сообщения.

- Выбирается эфемерный ключ $0 < k < Q$;
- Определяется $R = G^k(mod\ P)$;
- Находится $E = h(m||R)$, где $||$ - конкатенация сообщений;

- Вычисляется $S = (k + xE)(mod\ Q)$.

Подписью является пара (E, S).

2. Проверка подписи.

   - Вычисление $R' = G^S \cdot Y^{-E}(mod\ P)$;
   - Вычисление $E' = h(m||R')$;
   - Подпись корректна если $E' = E$.

Если мы сравним алгоритм DSA и подпись Шнорра то сразу увидим, что процесс проверки подписи значительно проще в алгоритме Шнорра и процесс становления подписи более криптостоик, так как используется конкатенация сообщения и значения R, что обеспечит различные подписи у одинаковых сообщений.

Рассмотрим пример работы подписи Шнорра.

Пример 3.4.1 $(P, Q, G) = (607, 601, 101)$

$x = 3$ - секретный ключ;

открытый ключ $Y = G^x(mod\ P) = 601^3(mod\ 607) = 391$.

1. *Подпись сообщения.*

   - *Эфемерный ключ* $k = 65$
   - $R = G^k(mod\ P) = 601^{65}(mod\ 607) = 223$
   - *Пусть* $E = h(m||R) = 93$
   - $S = (k + xE)(mod\ Q) = (65 + 3 \cdot 93)(mod\ 101) = 41$

   Подписью является пара $(93, 41)$.

2. *Проверку подписи в данном примере произвести не получится так как посчитать* $E' = h(m||R')$ *пока мы не можем.*

Благодаря своей компактности подпись Шнорра очень часто используется в модулях с ограниченными вычислительными ресурсами при аутентификации, например в смарт-картах - проходках. Рассмотрим пример как работает подпись Шнорра в смарт-картах при аутентификации.

Считыватель карты - автомат обладает копией открытого ключа $Y = G^x(mod\ P)$. На карточке записан секретный ключ x. Карточка все время генерирует эфемерные

ключи и вычисляет $R = G^k(mod\ P)$.

Когда смарт-карта помещается в считывающее устройство, она передает считывающему устройству одно из этих значений. Считыватель выдает сообщение $E = h(m\ R)$, карта в ответ вычисляет и передает $S = (k + xE)(mod\ Q)$.

После этого автомат уже проводит проверку подписи, вычисляя R' и E'. Схематически мы можем представить это как три шага, обозначив автомат за А, а карту за К.

1. $K \rightarrow A$ передает $R = G^k(mod\ P)$
2. $A \rightarrow K$ передает $E = h(m||R)$
3. $K \rightarrow A$ передает подпись $S = (k + xE)(mod\ Q)$. После этого пара (E, S) есть у автомата и он производит проверку подписи.

Соответственно процесс аутентификации происходит в три шага

Обращение → запрос → ответ.

Благодаря своей компактности подпись Шнорра широко применяется в BitCoin.

Если подпись ECDSA (DSA на эллиптических кривых) занимает 70 байт, то подпись Шнорр всего 64 байта, разница в 6 байт дает большой выигрыш, обеспечивая уменьшение размера всего блока данных и тем самым улучшая приватность.

4. ХЭШ-ФУНКЦИИ, ОСНОВНЫЕ ПОНЯТИЯ

Для начала знакомства с хэш-функциями давайте введем определение.

Определение 4.1.1 ***Хэш-функция (h- функция)*** *- функция, определенная на битовых строках произвольной длины со значениями в битовых строках фиксированной длины. Ее значение называется хэш-кодом (хэш- значением), или просто хэш.*

Мы будем обозначать хэш-значение сообщения m, как h(m).

Хэш-функции, в настоящее время, очень широко применяются повсеместно. Они помогают искать данные и хранить их, занимая меньше места. Когда мы ищем какую-то информацию в поисковых системах, то поиск идет как раз по хэш-значениям.

Например, если мы вводим запрос в поисковую строку, то вычисляется хэш-значение от вводимых нами данных, далее в системе ищутся данные с тем-же хэш-значением и при нахождении нам выдается ответ. Поэтому хэш-функции лежат в основе многих информационных процессов.

Криптографическая хэш-функция немного отличается от обыкновенных хэш-функций и должна удовлетворять ряду критериев:

1. Защищенность от восстановления прообразов: по Y из множества значений хэш-функции (МЗХФ) невозможно подобрать сообщение m из области определения (ОО). То есть вычислить Y: h (m) = Y, зная h(m) невозможно.

2. Защищенность от повторений (коллизии 1-го рода): не существует сообщения m не равного m': h (m) = h (m')

3. Защищенность от вторых прообразов (коллизии 2-го рода): по данному m невозможно найти m'≠ m: h (m) = h (m').

Давайте рассмотрим к каким проблемам может привести нарушение любого из этих критериев. Рассматривать это мы будем на примере создания подписи с помощью алгоритма RSA. Вспомним процесс создания подписи алгоритмом RSA.

Процесс создания подписи.

1. m сообщение, которое необходимо подписать. Вычисляется H= h (m);
2. применяя алгоритма RSA получаем подпись $S = H^d \pmod N$, где (d, p, q) – закрытый ключ RSA. Получаем: (m, S);
3. для проверки подписи вычисляется H': $H' = S^E \pmod N$, где (E, N) - открытый ключ и вычисляется H=h(m);
4. если H' = H то подпись достоверна.

При нарушении любого из перечисленных ранее критериев, при постановки подписи, будут возникать проблемы, которые Ева может использовать. Разберем на примерах, что будет происходить.

1. Отсутствует защищенность от восстановления прообразов, то есть возможно найти сообщение по данному значению хэш-функции:

   - Ева вычисляет H’ = S^E (mod N);
   - Вычисляет m = h^{-1}(H’);
   - У Евы есть (m, S).

2. Отсутствие защищенности от повторений – коллизии 1-го рода, то есть возможно найти два сообщения с одним и тем же значением хэш- функции:

   - Отправитель создает два разных сообщения m и m’ с одинаковым хэш- значением: h (m) = h (m’);
   - Подписывает m – (m, S);
   - Отказывается от подписи, утверждая, что подписывалось сообщение m’.

3. Отсутствие защищенности от вторых прообразов – коллизии 2-го рода, то есть, по данному сообщению возможно найти другое сообщение с тем же значением хэш-функции:

   - Ева получила (m, S);
   - Ева находит m’ =m: h (m) = h (m’);
   - Ставит подпись на сообщение m’ вместо m: (m’, S).

Существование коллизий объясняется парадоксом дней рождений.

5. СТРУКТУРА МЕРКЛА – ДАМГАРДА (ДАМГОРА)

Прежде чем знакомится с конкретными алгоритмами, рассмотрим одну структуру, лежащую в основе большинства криптографических хэш-функций. Это структура Меркла – Дамгарда (Дамгора).

Впервые она была предложена в 1979 году Ральфом Мерклом, позже выяснилось, что независимо от него такую же структуру предложил Иван Дамгор.

Суть структуры заключается в итеративном процессе последовательных преобразований. На вход каждой итерации поступает блок исходного текста и выход

предыдущей итерации.

Мы помним, что хеш-функция должна преобразовывать входные данные произвольной длины в выходное сообщение фиксированной длины. Меркл и Дамгард предложили добиться этого путём разбиения входного сообщения на одинаковые по размеру блоки, и последовательной обработкой этих блоков односторонней функцией сжатия. Функция сжатия может быть либо специально разработана для хеширования, либо представлять собой функцию блочного шифрования. Позже мы познакомимся и с той и с другой.

Наиболее известными алгоритмами, использующими структуру Меркла-Дамгарда (МД), являются MD5, SHA-1 и SHA-2, сегодня мы с ними познакомимся.

Более наглядно работа структуры МД представлена на рисунке 8.

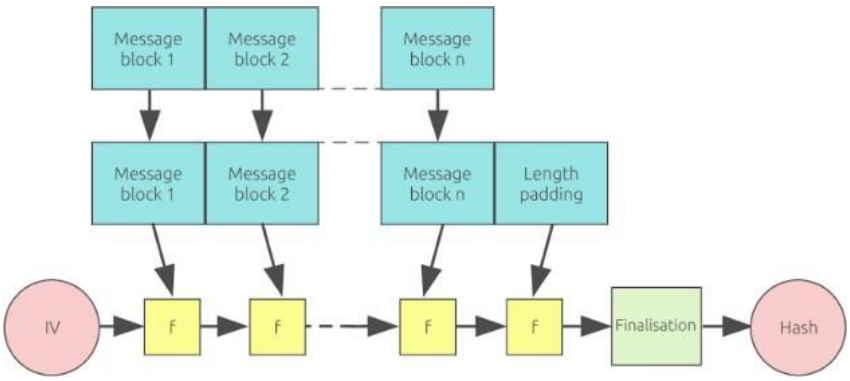

Рис. 8: Структура МД

Разберем ее работу по шагам:

1. На первом шаге мы разбиваем данные на блоки фиксированной длины;
2. После этого создается дополнительный блок, в который записывается - какой длины было сообщение;

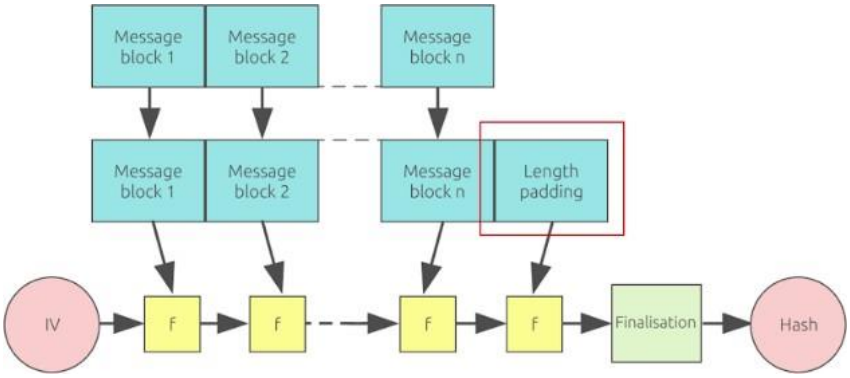

Рис. 9: Добавочный блок

3. После этого начинается сжатие с помощью любой односторонней сжимающей функции. Для начала в функцию сжатия поступает инициализирующий вектор IV и первый блок;

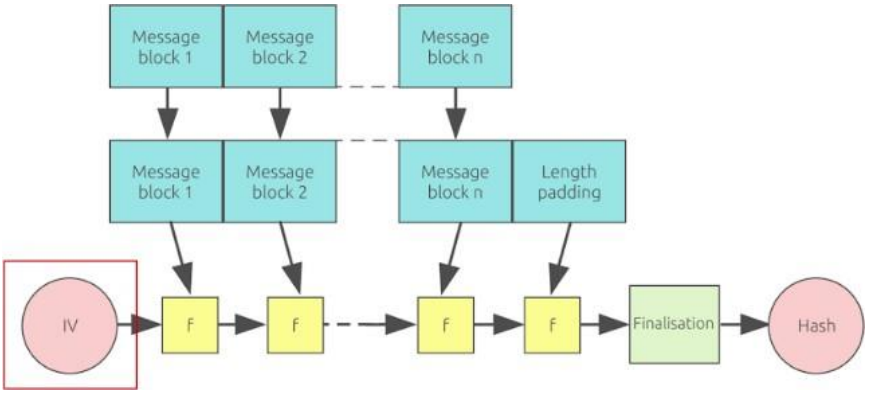

Рис. 10: Действие с инициализирующим вектором

4. На каждом шаге в сжимающую функцию поступает результат предыдущего шага и соответствующий блок сжимаемого текста;

5. На выходе получается хэш - значение нужной длины.

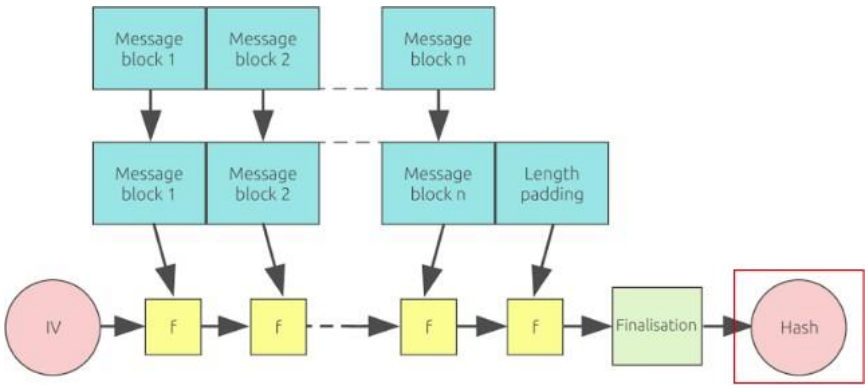

Рис. 11: Хэш-значение на выходе из структуры МД

Рассмотрим наиболее известные хэш-функции, использующие эту структуру.

5.СЕМЕЙСТВО ХЭШ-ФУНКЦИЙ MD

Семейство хэш-функций MD было предложено Роном Риверстом. Наибольшую популярность получила хэш-функция MD4 и ее улучшенная версия хэш-функция MD5. MD обозначает *Message Digest*, MD4 и MD5 выдают 128- битовое хэш-значение.

При разработке алгоритма MD4 учитывались следующие критерии:

- **Безопасность.** Вычислительно невозможно найти два сообщения с одинаковым хэш-значением. Самым эффективным является полный перебор.
- **Прямая безопасность.** Безопасность MD4 не основывается на каких- либо «трудно решаемых» задачах, например, разложение на множители.
- **Скорость.** MD4 подходит для высокоскоростных программных решений, так как основано на простом наборе битовых манипуляций с 32- битовыми операндами.
- **Простота и компактность.** MD4 проста, насколько это возможно, так как не содержит сложных программных модулей.
- **Архитектура.** MD4 оптимизирован для микропроцессорной архитектуры и легко изменяем.

Так как алгоритм MD5 является улучшенной версией алгоритма MD4, то ему присущи

все характеристики которые были заявлены Роном Риверстом при создании MD4. Сейчас мы познакомимся с алгоритмом MD5 и немного рассмотрим алгоритм MD4.

5.1 ХЭШ-ФУНКЦИЯ MD5

Хэш-функция MD5 выдает 128-битное значение. Работает MD5 с 512-битными блоками, каждый из которых разбивается на 16 32-битных подблока. Выходом алгоритма является набор из четырех 32-битных блоков, которые, при объединение, как раз дают 128-битное значение.

На первом шаге сообщение дополняется так, чтобы его длина была на 64 бита короче числа, кратного 512. Этим дополнением является единица, за которой до конца сообщения следуют нули. Далее к результату дополняется 64-битовое представление длины сообщения. Эти два действия гарантируют, что длина сообщения будет кратна 512 и что два разных сообщения всегда будут выглядеть по-разному после дополнения.

В алгоритме есть четыре 32-битных инициализирующие переменные, их называют **переменные сцепления**.

A=0x01234567
B=0x89abcdef
C=0xfedcba98
D=0x76543210

Мы помним, что алгоритм MD5 (как и MD4) работает со структурой Меркла-Дамгарда, переменные сцепления мы видим на рисунке 12.

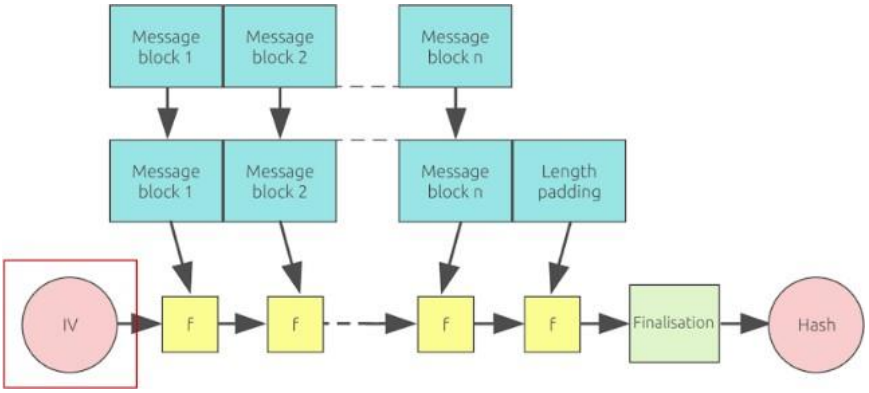

Рис. 12: Переменные сцепления

Переменные сцепления переписываются в другие переменные: A в a, B в b C в c, D в d.

Главный цикл состоит из четырех этапов (MD4 их было три). На каждом этапе происходит 16 операций.

1. каждая операция производится над тремя (из четырех) переменных и на выходе получает 32-битное значение. Полученное значение XOR-ся с четвертой переменной;
2. далее идет XOR с подблоком текста;
3. позже идет XOR со специальной константой;
4. циклический сдвиг;
5. XOR с одной из переменных a,b,c или d;
6. полученный результат записывается в одну из переменных a, b, c или d.

Более наглядно это представлено на рисунке 13.

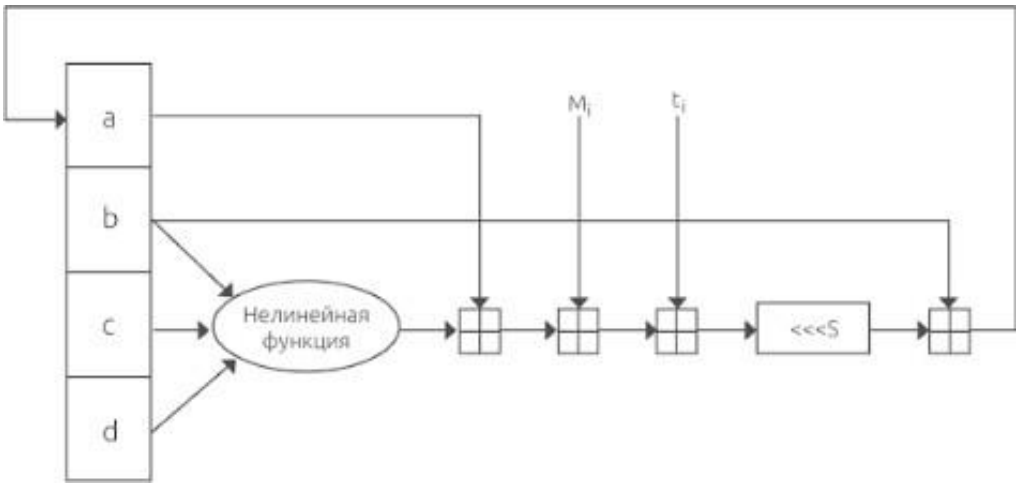

Рис. 13: Структура этапа MD5

Сама функция сжатия использует четыре преобразования. Функции представлены на рисунке 14.

$$F(X,Y,Z) = (X \wedge Y) \vee ((\neg X) \wedge Z)$$
$$G(X,Y,Z) = (X \wedge Z) \vee (Y \wedge (\neg Z))$$
$$H(X,Y,Z) = X \oplus Y \oplus Z$$
$$I(X,Y,Z) = Y \oplus (X \vee (\neg Z))$$

($\oplus$ – это XOR, $\wedge$ – AND, $\vee$ – OR, а $\neg$ – NOT)

Рис. 14: Функции алгоритма MD5

Запишем операции математически:

FF(a,b,c,d,m_j,s,t_i) обозначает, что a=b $\oplus((a \oplus F(b, c, d) \oplus m_j \oplus t_i) <<< s)$;

GG(a,b,c,d,m_j,s,t_i) обозначает, что a=b $\oplus((a \oplus G(b, c, d) \oplus m_j \oplus t_i) <<< s)$;

HH(a,b,c,d,m_j,s,t_i) обозначает, что a=b $\oplus((a \oplus H(b, c, d) \oplus m_j \oplus t_i) <<< s)$;

II(a,b,c,d,m_j,s,t_i) обозначает, что a=b $\oplus((a \oplus I(b, c, d) \oplus m_j \oplus t_i) <<< s)$;

где F(b,c,d), G(b,c,d), H(b,c,d), I(b,c,d) – введенные ранее функции сжатия, m_j подблок открытого текста, t_i константа, равная целой части 2^{32}abs(sin(i)), где i измеряется в радианах, s циклический сдвиг, прописанный для каждого шага.

Как мы видим, на каждом этапе происходит 16 шагов, при этом на каждом шаге используется своя константа и свой циклический сдвиг. За один этап проходят все подблоки хэшируемого текста, при этом на каждом этапе порядок работы с подблоками различен.

При сравнении MD4 и MD5 можно заметить:

1. Добавился четвертый этап;
2. В каждом действие используется уникальная константа;
3. Функция G на втором этапе была изменена;
4. Каждое действие прибавляется к результату предыдущего этапа, что обеспечивает более быстрый лавинный эффект;
5. Изменился порядок, в котором используются подблоки сообщения на этапах 2 и 3, что делает их менее похожими;

6. Циклический сдвиг был оптимизирован для увеличения лавинного эффекта.

В настоящее время существует ряд успешных атак на MD5 и MD4, но все равно MD5 широко применяется для хранения паролей в различных системах.

6. СЕМЕЙСТВО ХЭШ-ФУНКЦИЙ SHA

Семейство хэш-функций SHA (Secure Hash Algorithm) состоит из трех хэш-функций (SHA1, SHA2, SHA3), стандартизированных NIST (The National Institute of Standards and Technology). На этой лекции мы познакомимся с работой алгоритма SHA1 и немного поговорим о создании алгоритма SHA3.

6.1 ХЭШ-ФУНКЦИЯ SHA1

Хэш-функция SHA1 была создана в 1995 году и выдает 160-битное значение. Идеи, использованные в SHA1, очень схожи с идеями, применяемыми в MD4-MD5. Познакомимся с алгоритмом более подробно.

В алгоритме используется пять инициализирующих 32-битных вектора:

A=0x674542301

B=0xEFCDAB89

C=0x98BADCFE

D=0x10325476

E=0xC3D2E1F0

Разбиение сообщения на блоки происходит также как и в MD5, на блоки по 512 бит каждый. Переменные a, b, c, d, e, инициализируются значениями A, B, C, D, E, соответственно. В алгоритме происходит четыре этапа, по 20 итерация в каждом.

Блок сообщения преобразуется из 16 32-битовых слов m_i в 80 32-битовых слов W_j по следующему правилу:

$W_t=m_t$, при $0 < t < 15$

$W_t=W_{t-3}$ XOR W_{t-8} XOR W_{t-14} XOR $W_{t-16} < < < 1$, при $16 < t < 79$.

Где < < < - обозначает циклический сдвиг, а t номер итерации/шага.

В алгоритме используется следующие 4 функции, представленные на рисунке 15.

$f_t(X,Y,Z) = (X \wedge Y) \vee ((\neg X) \wedge Z)$, для t = 0 до 19

$f_t(X,Y,Z) = X \oplus Y \oplus Z$, для t = 20 до 39

$f_t(X,Y,Z) = (X \wedge Y) \vee (X \wedge Z) \vee (Y \wedge Z)$, для t = 40 до 59

$f_t(X,Y,Z) = X \oplus Y \oplus Z$, для t = 60 до 79

Рис. 15: Нелинейные функции алгоритма SHA1

Также в алгоритме используются 4 константы.

K_t= 0x5A827999, 0 < t <19;

K_t= 0x6ED9EBA1, 20 <t <39;

K_t= 0x8F1BBCDC, 40 <t < 59;

K_t= 0xCA62C1D6, 60 < t < 79.

Схема работы одного шага алгоритма представлена на рисунке 16.

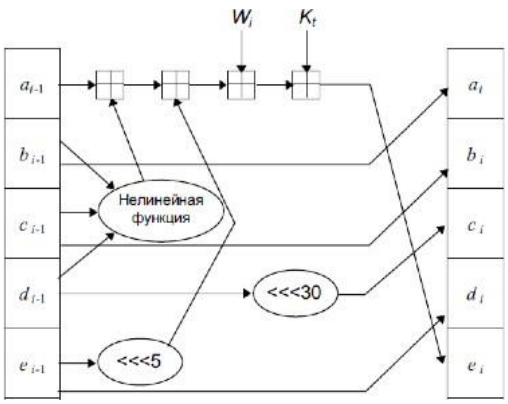

Рис. 16: Шаг алгоритма SHA1

Как мы видим на схеме, алгоритм SHA1 очень схож с алгоритмом MD5, но с 2005 года появилось довольно много успешных атак, после которых был объявлен новый конкурс на создание алгоритма SHA3.

6.2 АТАКИ НА SHA1 И КОНКУРС SHA3

В январе 2005 года Vincent Rijmen и Elisabeth Oswald опубликовали статью об атаке на усечённую версию SHA1 (53 раунда вместо 80), которая позволяет находить коллизии меньше, чем за 2^{80} операций.

Далее Xiaoyun Wang, Yiqun Lisa Yin, Hongbo Yu представили атаку на полноценный SHA1,

которая требует менее 2^{69} операций. В этой же статье была опубликована атаку на 58 раундов SHA1, которая позволяет находить коллизии за 2^{33} операций.

В августе 2005 года на конференции CRYPTO2005 ими же была представлена улучшенную версию атаки на полную версию алгоритма с вычислительной сложностью в 2^{63} операций.

Позже была представлена усовершенствованная версия атаки на SHA1 с вычислительной сложностью около 2^{35} операций.

Теоретически SHA1 считается взломанным (количество вычислительных операций сокращено в 2^{80-63} = 131 000 раз), на практике подобный взлом неосуществим, так как займёт пять миллиардов лет.

Так как алгоритм SHA1 оказался «взломан», NIST отказался от использования SHA1 в цифровых подписях.

2 ноября 2007 года NIST анонсировало конкурс по разработке нового алгоритма SHA3.

Здесь мы видим список хэш-функций вышедших в полуфинал, все они достаточно интересны.

Hash Name	Principal Submitter
BLAKE	Jean-Philippe Aumasson
Blue Midnight Wish	Svein Johan Knapskog
CubeHash	Daniel J. Bernstein
ECHO	Henri Gilbert
Fugue	Charanjit S. Jutla
Grøstl	Lars R. Knudsen
Hamsi	Özgül Küçük
JH	Hongjun Wu
Keccak	The Keccak Team
Luffa	Dai Watanabe
Shabal	Jean-François Misarsky
SHAvite-3	Orr Dunkelman
SIMD	Gaëtan Leurent
Skein	Bruce Schneier

Рис. 17: Полуфиналисты конкурса SHA3

В финал вышли алгоритмы:

1. BLAKE;
2. Grøstl;
3. JH;
4. Keccak;
5. Skein.

Победителем стал алгоритм Keccak, который предложил новую структуру под названием «губка».

7. ЗАКЛЮЧЕНИЕ

На данной лекции мы познакомились с протоколом распределения ключей Диффи-Хеллмана, который послужил началом эры асимметричного шифрования, рассмотрели его работу и изучили атаку "человек посередине". Познакомились с электронной подписью, применили алгоритм RSA не для шифрования, а для подписания данных, также узнали алгоритм DSA, являющийся стандартом шифрования в США, и подпись Шнорра - наиболее компактную подпись из всех существующих. На данной лекции мы также познакомились с хэш-функциями. Рассмотрели основные принципы работы криптографических хэш-функций, ввели понятие коллизий 1-го и 2-го рода. Рассмотрели структуру Меркла-Дамгарда(Дамгора), познакомились с наиболее известными хэш-функциями, такими как MD4, MD5, SHA1.