Глава 18
Однонаправленные хэш-функции

18.1 Основы

Однонаправленная функция $H(M)$ применяется к сообщению произвольной длины M и возвращает значение фиксированной длины h.

$h = H(M)$, где h имеет длину m

Многие функции позволяют вычислять значение фиксированной длины по входным данным произвольной длины, но у однонаправленных хэш-функций есть дополнительные свойства, делающие их однонаправленными [1065]:

Зная M, легко вычислить h.

Зная H, трудно определить M, для которого $H(M)=h$.

Зная M, трудно определить другое сообщение, M', для которого $H(M)= H(M')$.

Если бы Мэллори умел делать трудные вещи, он смог бы разрушить безопасность любого протокола, и с-пользующего однонаправленную хэш-функцию. Смысл однонаправленных хэш-функций и состоит в обеспеч е-нии для M уникального идентификатора ("отпечатка пальца"). Если Алиса подписала M с помощью алгоритма цифровой подписи на базе $H(M)$, а Боб может создать M', другое сообщение, отличное от M, для которого $H(M)= H(M')$, то Боб сможет утверждать, что Алиса подписала M'.

В некоторых приложениях однонаправленности недостаточно, необходимо выполнение другого требования, называемого **устойчивостью к столкновениям**.

Должно быть трудно найти два случайных сообщения, M и M', для которых $H(M)= H(M')$.

Помните вскрытие методом дня рождения из раздела 7.4? Оно основано не на поиске другого сообщения M', для которого $H(M)= H(M')$, а на поиске двух случайных сообщений, M и M', для которых $H(M)= H(M')$.

Следующий протокол, впервые описанный Гидеоном Ювалом (Gideon Yuval) [1635], показывает, как, если предыдущее требование не выполняется, Алиса может использовать вскрытие методом дня рождения для обм а-на Боба.

(1) Алиса готовит две версии контракта: одну, выгодную для Боба, и другую, приводящую его к банкротству

(2) Алиса вносит несколько незначительных изменений в каждый документ и вычисляет хэш-функции. (Этими изменениями могут быть действия, подобные следующим: замена ПРОБЕЛА комбинацией ПРО-БЕЛ-ЗАБОЙ-ПРОБЕЛ, вставка одного-двух пробелов перед возвратом каретки, и т.д. Делая или не делая по одному изменению в каждой из 32 строк, Алиса может легко получить 2^{32} различных документов.)

(3) Алиса сравнивает хэш-значения для каждого изменения в каждом из двух документов, разыскивая пару, для которой эти значения совпадают. (Если выходом хэш-функции является всего лишь 64-разрядное зн а-чение, Алиса, как правило, сможет найти совпадающую пару сравнив 2^{32} версий каждого документа.) Она восстанавливает два документа, дающих одинаковое хэш-значение.

(4) Алиса получает подписанную Бобом выгодную для него версию контракта, используя протокол, в котором он подписывает только хэш-значение.

(5) Спустя некоторое время Алиса подменяет контракт, подписанный Бобом, другим, который он не подпис ы-вал. Теперь она может убедить арбитра в том, что Боб подписал другой контракт.

Это заметная проблема. (Одним из советов является внесение косметических исправлений в подписываемый документ.)

При возможности успешного вскрытия методом дня рождения, могут применяться и другие способы вскр ы-тия. Например, противник может посылать системе автоматического контроля (может быть спутниковой) сл у-чайные строки сообщений со случайными строками подписей. В конце концов подпись под одним из этих сл у-чайных сообщений окажется правильной. Враг не сможет узнать, к чему приведет эта команда, но, если его единственной целью является вмешательство в работу спутника, он своего добьется.

Длины однонаправленных хэш-функций

64-битовые хэш-функции слишком малы, чтобы противостоять вскрытию методом дня рождения. Более практичны однонаправленные хэш-функции, выдающие 128-битовые хэш-значения. При этом, чтобы найти два

документа с одинаковыми хэш-значениями, для вскрытия методом дня рождения придется хэшировать 2^{64} случайных документов, что, впрочем, недостаточно, если нужна длительная безопасность. NIST в своем Стандарте безопасного хэширования (Secure Hash Standard, SHS), использует 160-битовое хэш-значение. Это еще сильнее усложняет вскрытие методом дня рождения, для которого понадобится 2^{80} хэширований.

Для удлинения хэн-значений, выдаваемых конкретной хэш-функцией, был предложен следующий метод.

(1) Для сообщения с помощью одной из упомянутых в этой книге однонаправленных хэш-функций генерируется хэш-значение.

(2) Хэш значение добавляется к сообщению.

(3) Генерируется хэш-значение объединения сообщения и хэш-значения этапа (1).

(4) Создается большее хэш-значение, состоящее из объединения хэш-значения этапа (1) и хэш-значения этапа (3).

(5) Этапы (1)-(4) повторяются нужное количество раз для обеспечения требуемой длины хэш-значения.

Хотя никогда не была доказана безопасность или небезопасность этого метода, уряд людей этот метод вызывает определенные сомнения [1262,859].

Обзор однонаправленных хэш-функций

Не легко построить функцию, вход которой имеет произвольный размер, а тем более сделаьть ее однонаправленной. В реальном мире однонаправленные хэш-функции строятся на идее **функции сжатия**. Такая однонаправленная функция выдает хэш-значение длины n при заданных входных данных большей длины m [1069, 414]. Входами функции сжатия являются блок сообщения и выход предыдущего блока текста (см. 17-й). Выход представляет собой хэш-значение всех блоков до этого момента. То есть, хэш-значение блока M_i равно

$h_i = f(M_i, h_{i-1})$

Это хэш-значение вместе со следующим блоком сообщения становится следующим входом функции сжатия. Хэш-значением всего сообщения является хэш-значение последнего блока.

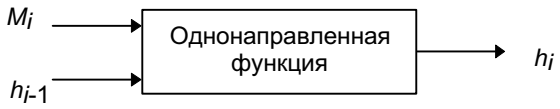


Рис. 18-1. Однонаправленная функция

Хэшируемый вход должен каким-то способом содержать бинарное представление длины всего сообщения. Таким образом преодолевается потенциальная проблема, вызванная тем, что сообщения различной длины могут давать одно и то же хэш-значение [1069, 414]. Иногда такой метод называется **MD-усилением** [930].

Различные исследователи выдвигали предположения, что если функция сжатия безопасна, то этот метод хэширования исходных данных произвольной длины также безопасен - но ничего не было доказано [1138, 1070, 414].

На тему проектирования однонаправленных хэш-функций написано много. Более подробную математическую информацию можно найти [1028, 793, 791, 1138, 1069, 414, 91, 858, 1264]. Возможно самым толковой интерпретацией однонаправленных хэш-функций являются тезисы Барта Пренела (Bart Preneel) [1262].

18.2 Snefru

Snefru - это однонаправленная хэш-функция, разработанная Ральфом Мерклом [1070]. (Snefru, также как Khufu и Khafre, был египетским фараоном.) Snefru хэширует сообщения произвольной длины, превращая их в 128-битовые 256-битовые значения.

Сначала собщение разбивается на кусочки длиной по 512-m. (Переменная m является длитной хэш-значения.) Если выход - это 128-битовое значение, то длина кусочков равна 384 битам, а если выход - 128-битовое значение, то длина кусочков - 256 битов.

Сердцем алгоритма служит функция H, хэширующая 512-битовое значение в m-битовое. Первые m битов выхода H являются хэш-значением блока, остальные отбрасываются. Следующий блок добавляется к хэш-значению предыдущего блока и снова хэшируется. (К первоначальному блоку добавляется строка нулей.) После последнего блока (если сообщение состоит не из целого числа блоков, последний блок дополняется нулями) первые m битов добавляются к бинарному представлению длины сообщения и хэшируются последний раз.

Функция H основывается на E, обратимой функции блочного шифрования, работающей с 512 битовыми

блоками. H - это последние *m* битов выхода E, объединенные посредством XOR с первыми *m* битами входа E.

Безопасность Snefru опирается на функцию E, которая рандомизирует данные за несколько проходов. Каждый проход состоит из 64 рандомизирующих этапов. В каждом этапе в качестве входа S-блока используется другой байт данных. Выходное слово подвергается операции XOR с двумя соседними словами сообщения. Построение S-блоков аналогично построению S-блоков в Khafre (см. раздел 13.7). Кроме того, выполняется ряд циклических сдвигов. Оригинальный Snefru состоял из двух проходов.

Криптоанализ Snefru

Используя дифференциальный криптоанализ, Бихам и Шамир показали небезопасность двухпроходного Snefru (с 128-битовым хэш-значением) [172]. Их способ вскрытия за несколько минут обнаруживает пару сообщений с одинаковым хэш-значением.

Для 128-битового Snefru их вскрытия работают лучше, чем вскрытие грубой силой для четырех и менее проходов. Вскрытие Snefru методом дня рождения требует 2^{64} операций; дифференциальный криптоанализ может найти пару сообщений с одинаковым хэш-значением за $2^{28.5}$ операций для трехпроходного Snefru и за $2^{44.5}$ операций для четырехпроходного Snefru. Нахождение сообщения, хэш-значение которого совпадает с заданным, при использовании грубой силы требует 2^{128} операций, при дифференциальном криптоанализе для этого нужно 2^{56} операций для трехпроходного и 2^{88} операций для четырехпроходного Snefru.

Хотя Бихам и Шамир не анализировали 256-битовые хэш-значения, они провели анализ вплоть до 224-битовых хэш-значений. В сравнении с вскрытием методом дня рождения, требующим 2^{112} операций они могут найти сообщения с одинаковым хэш-значением за $2^{12.5}$ операций для двухпроходного Snefru, за 2^{33} операций для трехпроходного Snefru и за для 2^{81} операций для четырехпроходного Snefru.

В настоящее время Меркл рекомендует использовать Snefru по крайней мере с восемью проходами [1073]. Однако с таким количеством проходов алгоритм становится намного медленнее, чем MD5 или SHA.

18.3 *N*-хэш

N-хэш - это алгоритм, придуманный в 1990 году исследователями Nippon Telephone and Telegraph, теми же людьми, которые изобрели FEAL [1105, 1106]. *N*-хэш использует 128-битовые блоки сообщения, сложную рандомизирующую функцию, похожую на FEAL, и выдает 128-битовое хэш-значение.

Хэш-значение каждого 128-битового блока является функцией блока и хэш-значения предыдущего блока.

$H_0 = I$, где I - случайное начальное значение

$H_i = g(M_i, H_{i-1}) \oplus M_i \oplus H_{i-1}$

Хэш-значение всего сообщения представляет собой хэш-значение последнего блока сообщения. Случайное начальное значение I может быть любым числом, определенным пользователем (даже одними нулями).

Функция g достаточно сложна. Схема алгоритма приведена на 16-й. Сначала переставляются левая и правая 64-битовые половины 128-битового хэш-значения предыдущего блока H_{i-1}, а затем выполняется XOR с повторяющимся шаблоном (128-битовым) и XOR с текущим блоком сообщения M_i. Далее это значение каскадно преобразуется в N (на рисунках N= 8) стадий обработки. Другим входом стадии обработки является предыдущее хэш-значение, подвергнутое XOR с одной из восьми двоичных констант.

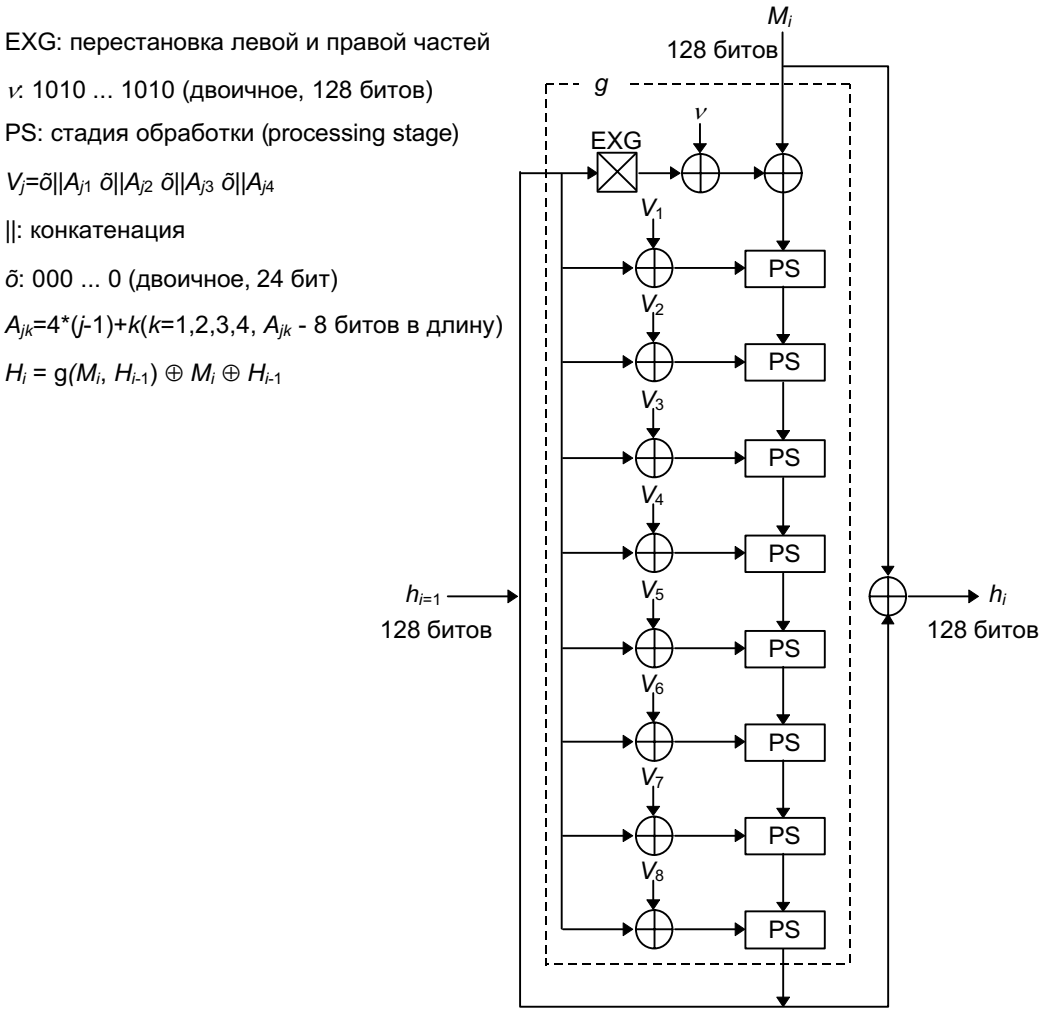


Рис. 18-2. Схема *N*-хэш.

Одна стадия обработки показана на 15-й. Блок сообщения разбивается на четыре 32-битовых значения . Предыдущее хэш-значение также разбивается на четыре 32-битовых значения . Функция f представлена на 14th. Функции S_0 и S_1 те же самые, что и в FEAL.

$S_0(a,b)$ = циклический сдвиг влево на два бита ((a + b) mod 256)

$S_1(a,b)$ = циклический сдвиг влево на два бита((a + b + 1) mod 256)

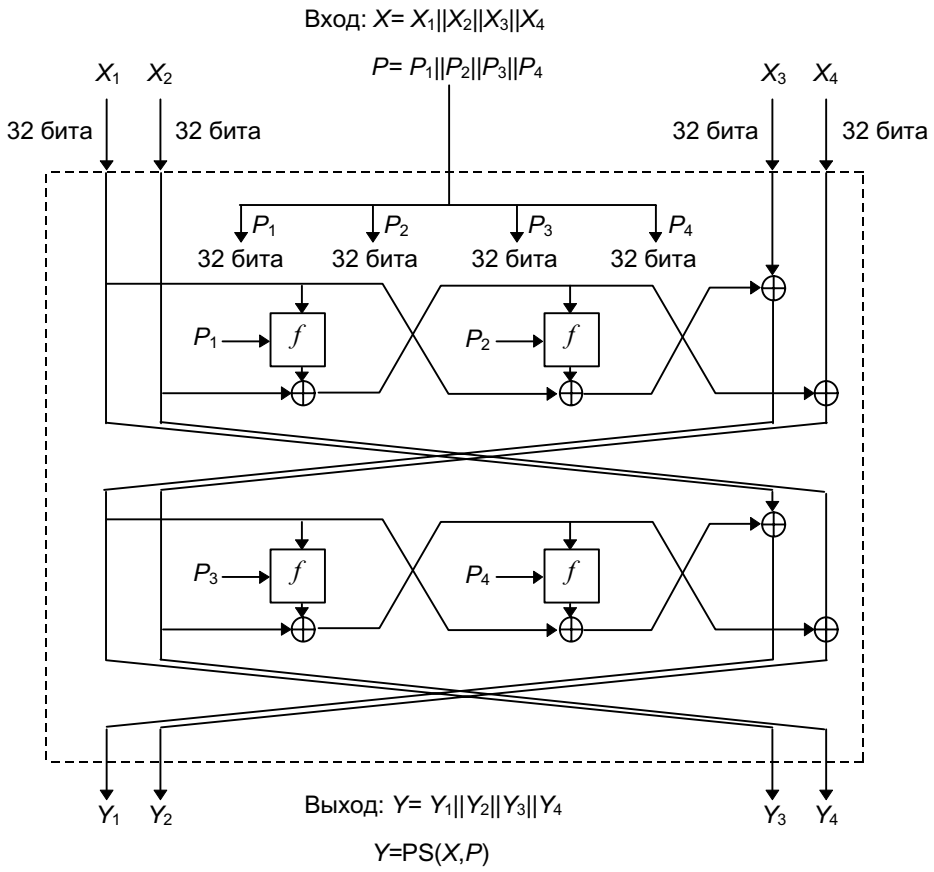
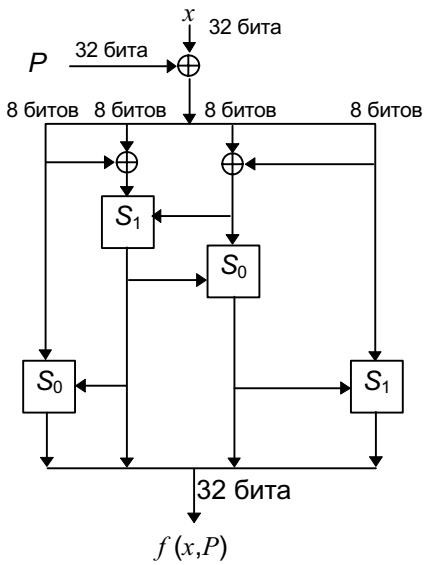


Рис. 18-3. Одна стадия обработки *N*-хэш.

Выход одной стадии обработки становится входом следующей стадии обработки . После последней стадии обработки выполняется XOR выхода с M_i и H_{i-1}, а затем к хэшированию готов следующий блок .



$Y=S_0(X_1,X_2)$=Rot2((X_1+X_2) mod 256)
$Y=S_1(X_1,X_2)$=Rot2((X_1+X_2+1) mod 256)
Y: выходные 8 битов, X_1,X_2 (8 битов): входы
Rot2(*Y*): циклический сдвиг влево на 2 бита 8-битовых данных *Y*

Рис. 18-4. Функция *f*.

Криптоанализ N-хэш

Берт ден Боер (Bert den Boer) открыл способ создавать столкновения в функции этапа *N*-хэш [1262]. Бихам и Шамир применили дифференциальный криптоанализ для вскрытия 6-этапной *N*-хэш [169, 172]. Конкретное выполненное ими вскрытие (конечно же, могли быть и другие) работает для любого *N*, делящегося на 3, и эффективнее вскрытия методом дня рождения для любого *N*, меньшего 15.

То же самое вскрытие может обнаруживать пары сообщений с одинаковым хэш-значением для 12-этапной *N*-хэш за 2^{56} операций (для вскрытия грубой силой нужно 2^{64} операций). *N*-хэш с 15 этапами безопасна по отношению к дифференциальному криптоанализу: для вскрытия потребуется 2^{72} операций.

Разработчики алгоритма рекомендуют использовать *N*-хэш не меньше, чем с 8 этапами [1106]. С учетом доказанной небезопасности *N*-хэш и FEAL (и ее скорости при 8 этапах) я рекомендую полностью отказаться от этого алгоритма.

18.4 MD4

MD4 - это однонаправленная хэш-функция, изобретенная Роном Ривестом [1318, 1319, 1321]. MD обозначает Message Digest (краткое изложение сообщения), алгоритм для входного сообщения выдает 128-битовое хэш-значение, или краткое изложение сообщения.

В [1319] Ривест описал цели, преследуемые им при разработке алгоритма:

Безопасность. Вычислительно невозможно найти два сообщения с одинаковым хэш-значением. Вскрытие грубой силой является самым эффективным.

Прямая безопасность. Безопасность MD4 не основывается на каких-либо допущениях, например, предположении о трудности разложения на множители.

Скорость. MD4 подходит для высокоскоростных программных реализаций. Она основана на простом наборе битовых манипуляций с 32-битовыми операндами.

Простота и компактность. MD4 проста, насколько это возможна, и не содержит больших структур данных или сложных программных модулей.

Удачна архитектура. MD4 оптимизирована для микропроцессорной архитектуры (особенно для микропроцессоров Intel), для более крупных и быстрых компьютеров можно выполнить любые необходимые изменения.

После первого появления алгоритма Берт ден Боер и Антон Босселаерс (Antoon Bosselaers) достигли успеха при криптоанализе последних двух из трех этапов алгоритма [202]. Ральфу Мерклу совершенно независимо удалось вскрыть первые два этапа [202]. Эли Бихам рассмотрел использование дифференциального криптоанализа против первых двух этапов MD4 [159]. Хотя все эти вскрытия не были распространены на полный алгоритм, Ривест усилил свою разработку. В результате появилась MD5.

18.5 MD5

MD5 - это улучшенная версия MD4 [1386, 1322]. Хотя она сложнее MD4, их схемы похожи, и результатом MD5 также является 128-битовое хэш-значение.

Описание MD5

После некоторой первоначальной обработки MD5 обрабатывает входной текст 512-битовыми блоками, разбитыми на 16 32-битовых подблоков. Выходом алгоритма является набор из четырех 32-битовых блоков, которые объединяются в единое 128-битовое хэш-значение.

Во первых, сообщение дополняется так, чтобы его длина была на 64 бита короче числа, кратного 512. Этим дополнением является 1, за которой вплоть до конца сообщения следует столько нулей, сколько нужно. Затем, к результату добавляется 64-битовое представление длины сообщения (истинной, до дополнения). Эти два действия служат для того, чтобы длина сообщения была кратна 512 битам (что требуется для оставшейся части алгоритма), и чтобы гарантировать, что разные сообщения не будут выглядеть одинаково после дополнения. Инициализируются четыре переменных:

A = 0x01234567

B = 0x89abcdef

C = 0xfedcba98

D = 0x76543210

Они называются **переменными сцепления**.

Теперь перейдем к основному циклу алгоритма. Этот цикл продолжается, пока не исчерпаются 512-битовые блоки сообщения.

Четыре переменных копируются в другие переменные: *A* в *a*, *B* в *b*, *C* в *c* и *D* в *d*.

Главный цикл состоит из четырех очень похожих этапов (у MD4 было только три этапа). На каждом этапе 16 раз используются различные операции. Каждая операция представляет собой нелинейную функцию над тремя из *a*, *b*, *c* и *d*. Затем она добавляет этот результат к четвертой переменной, подблоку текста и константе. Далее результат циклически сдвигается вправо на переменное число битов и добавляет результат к одной из переменных *a*, *b*, *c* и *d*. Наконец результат заменяет одну из переменных *a*, *b*, *c* и *d*. См. 13-й и 12-й. Существуют четыре нелинейных функции, используемые по одной в каждой операции (для каждого этапа - другая функция).

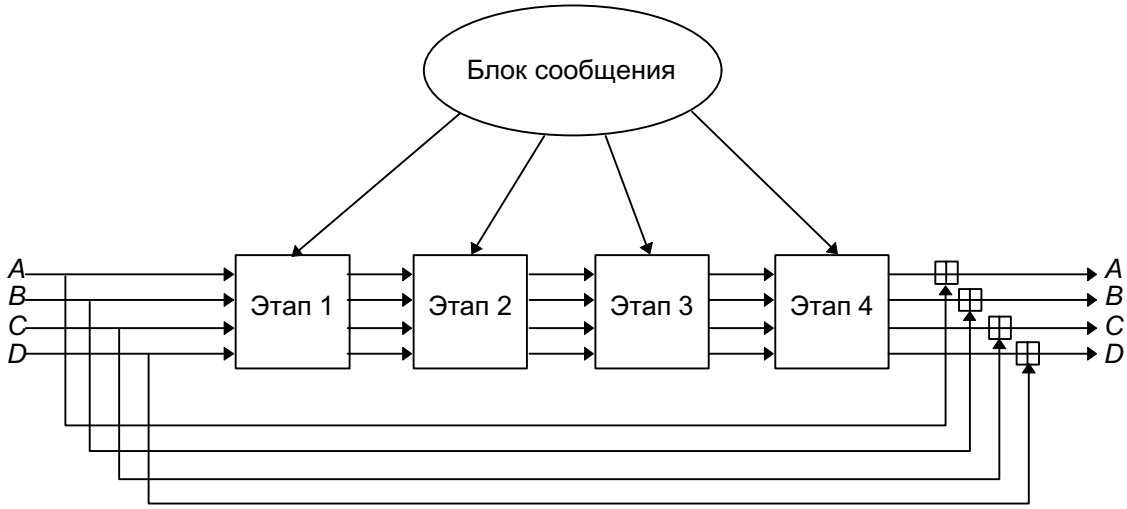


Рис. 18-5. Главный цикл MD5.

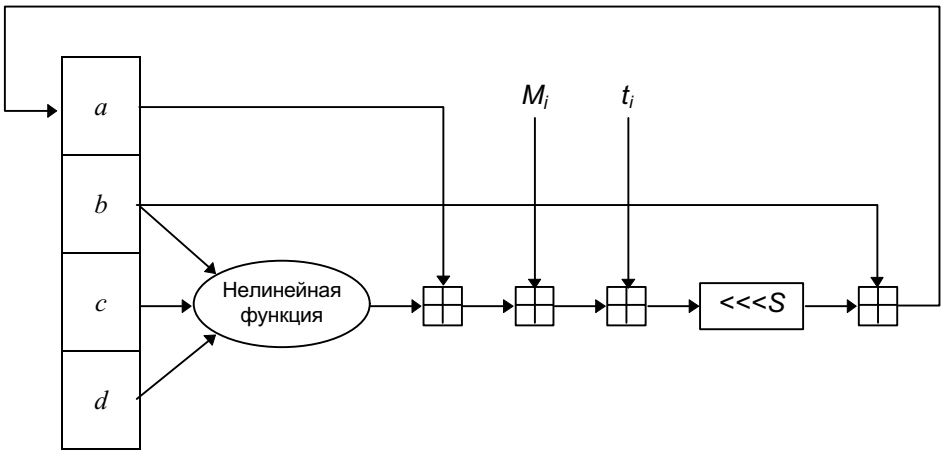


Рис. 18-6. Одна операция MD5.

$F(X,Y,Z) = (X \wedge Y) \vee ((\neg X) \wedge Z)$

$G(X,Y,Z) = (X \wedge Z) \vee (Y \wedge (\neg Z))$

$H(X,Y,Z) = X \oplus Y \oplus Z$

$I(X,Y,Z) = Y \oplus (X \vee (\neg Z))$

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

Эти функции спроектированы так, чтобы, если соответствующие биты *X*, *Y* и *Z* независимы и несмещены, каждый бит результата также был бы независимым и несмещенным. Функция F - это побитовое условие: если *X*, то *Y*, иначе *Z*. Функция H - побитовая операция четности.

Если M_j обозначает j-ый подблок сообщения (от 0 до 15), а $<<<s$ обозначает циклический сдвиг влево на *s*

битов, то используются следующие четыре операции:

FF(a,b,c,d,M_j,s,t_i) означает $a = b + ((a + F(b,c,d) + M_j + t_i) <<< s)$

GG(a,b,c,d,M_j,s,t_i) означает $a = b + ((a + G(b,c,d) + M_j + t_i) <<< s)$

HH(a,b,c,d,M_j,s,t_i) означает $a = b + ((a + H(b,c,d) + M_j + t_i) <<< s)$

II(a,b,c,d,M_j,s,t_i) означает $a = b + ((a + I(b,c,d) + M_j + t_i) <<< s)$

Четыре этапа (64 действия выглядят следующим образом):

Этап 1:

FF(a, b, c, d, M_0, 7, 0xd76aa478)

FF(d, a, b, c, M_1, 12, 0xe8c7b756)

FF(c, d, a, b, M_2, 17, 0x242070db)

FF(b, c, d, a, M_3, 22, 0xc1bdceee)

FF(a, b, c, d, M_4, 7, 0xf57c0faf)

FF(d, a, b, c, M_5, 12, 0x4787c62a)

FF(c, d, a, b, M_6, 17, 0xa8304613)

FF(b, c, d, a, M_7, 22, 0xfd469501)

FF(a, b, c, d, M_8, 7, 0x698098d8)

FF(d, a, b, c, M_9, 12, 0x8b44f7af)

FF(c, d, a, b, M_{10}, 17, 0xffff5bb1)

FF(b, c, d, a, M_{11}, 22, 0x895cd7be)

FF(a, b, c, d, M_{12}, 7, 0x6b901122)

FF(d, a, b, c, M_{13}, 12, 0xfd987193)

FF(c, d, a, b, M_{14}, 17, 0xa679438e)

FF(b, c, d, a, M_{15}, 22, 0x49b40821)

Этап 2:

GG(a, b, c, d, M_1, 5, 0xf61e2562)

GG(d, a, b, c, M_6, 9, 0xc040b340)

GG(c, d, a, b, M_{11}, 14, 0x265e5a51)

GG(b, c, d, a, M_0, 20, 0xe9b6c7aa)

GG(a, b, c, d, M_5, 5, 0xd62fl05d)

GG(d, a, b, c, M_{10}, 9, 0x02441453)

GG(c, d, a, b, M_{15}, 14, 0xd8ale681)

GG(b, c, d, a, M_4, 20, 0xe7d3fbc8)

GG(a, b, c, d, M_9, 5, 0x2,lelcde6)

GG(d, a, b, c, M_{14}, 9, 0xc33707d6)

GG(c, d, a, b, M_3, 14, 0xf4d50d87)

GG(b, c, d, a, M_8, 20, 0x455al4ed)

GG(a, b, c, d, M_{13}, 5, 0xa9e3e905)

GG(d, a, b, c, M_2, 9, 0xfcefa3f8)

GG(c, d, a, b, M_7, 14, 0x676f02d9)

GG(b, c, d, a, M_{12}, 20, 0x8d2a4c8a)

Этап 3:

HH(a, b, c, d, M_5, 4, 0xfffa3942)

HH(d, a, b, c, M_8, 11, 0x8771f681)

HH(c, d, a, b, M_{11}, 16, 0x6d9d6122)

HH(b, c, d, a, M_{14}, 23, 0xfde5380c)

HH(a, b, c, d, M_1, 4, 0xa4beea44)

HH(d, a, b, c, M_4, 11, 0x4bdecfa9)

HH(c, d, a, b, M_7, 16, 0xf6bb4b60)

HH(b, c, d, a, M_{10}, 23, 0xbebfbc70)

HH(a, b, c, d, M_{13}, 4, 0x289b7ec6)

HH(d, a, b, c, M_0, 11, 0xeaa127fa)

HH(c, d, a, b, M_3, 16, 0xd4ef3085)

HH(b, c, d, a, M_6, 23, 0x04881d05)

HH(a, b, c, d, M_9, 4, 0xd9d4d039)

HH(d, a, b, c, M_{12}, 11, 0xe6db99e5)

HH(c, d, a, b, M_{15}, 16, 0x1fa27cf8)

HH(b, c, d, a, M_2, 23, 0xc4ac5665)

Этап 4:

II(a, b, c, d, M_0, 6, 0xf4292244)

II(d, a, b, c, M_7, 10, 0x432aff97)

II(c, d, a, b, M_{14}, 15, 0xab9423a7)

II(b, c, d, a, M_5, 21, 0xfc93a039)

II(a, b, c, d, M_{12}, 6, 0x655b59c3)

II(d, a, b, c, M_3, 10, 0x8f0ccc92)

II(c, d, a, b, M_{10}, 15, 0xffeff47d)

II(b, c, d, a, M_1, 21, 0x85845ddl)

II(a, b, c, d, M_8, 6, 0x6fa87e4f)

II(d, a, b, c, M_{15}, 10, 0xfe2ce6e0)

II(c, d, a, b, M_6, 15, 0xa3014314)

II(b, c, d, a, M_{13}, 21, 0x4e081lal)

II(a, b, c, d, M_4, 6, 0xf7537e82)

II(d, a, b, c, M_{11}, 10, 0xbd3af235)

II(c, d, a, b, M_2, 15, 0x2ad7d2bb)

II(b, c, d, a, M_9, 21, 0xeb86d391)

Эти константы, t_i, выбирались следующим образом:

На i-ом этапе t_i является целой частью 2^{32}*abs(sin(i)), где i измеряется в радианах.

После всего этого a, b, c и d добавляются к A, B, C и D, соответственно, и алгоритм продолжается для следующего блока данных. Окончательным результатом служит объединение A, B, C и D.

Безопасность MD5

Рон Ривест привел следующие улучшения MD5 в сравнении с MD4 [1322]:

1. Добавился четвертый этап.
2. Теперь в каждом действии используется уникальная прибавляемая константа .

3. Функция G на этапе 2 с $((X \wedge Y) \vee (X \wedge Z) \vee (Y \wedge Z))$ была изменена на $(X \wedge Z) \vee (Y \wedge (\neg Z))$, чтобы сделать G менее симметричной.

4. Теперь каждое действие добавляется к результату предыдущего этапа. Это обеспечивает более быстрый лавинный эффект.

5. Изменился порядок, в котором использовались подблоки сообщения на этапах 2 и 3, чтобы сделать шаблоны менее похожими.

6. Значения циклического сдвига влево на каждом этапе были приближенно оптимизированы для ускорения лавинного эффекта. Четыре сдвига, используемые на каждом этапе, отличаются от значений, используемых на других этапах.

Том Берсон (Tom Berson) попытался применить дифференциальный криптоанализ к одному этапу MD5 [144], но его вскрытие не оказалось эффективным ни для одного из четырех этапов. Более успешное вскрытие ден Боера и Босселаерса, использующее функцию сжатия, привело к обнаружению столкновений в MD5 [203, 1331, 1336]. Само по себе это вскрытие невозможно для вскрытия MD5 в практических приложениях, оно не влияет и на использование MD5 в алгоритмах шифрования, подобных Luby-Rackoff (см. раздел 14.11). Успех этого вскрытия означает только, что одна из основных целей проектирования MD5- создать устойчивую к столкновениям функцию сжатия - не была достигнута. Хотя справедливо, что "кажется, что у функции сжатия есть слабое место, но это практически не влияет на безопасность хэш-функции " [1336], я отношусь к использованию MD5 очень осторожно.

18.6 MD2

MD2 - это другая 128-битовая однонаправленная хэш-функция, разработанная Роном Ривестом [801, 1335]. Она, вместе с MD5, используется в протоколах PEM (см. раздел 24.10). Безопасность MD2 опирается на случайную перестановку байтов. Эта перестановка фиксирована и зависит от разрядов π. S_0, S_1, S_2, . . . , S_{255} и являются перестановкой. Чтобы выполнить хэширование сообщения M:

(1) Дополните сообщение i байтами, значение i должно быть таким, чтобы длина полученного сообщения была кратна 16 байтам.

(2) Добавьте к сообщению 16 байтов контрольной суммы.

(3) Проинициализируйте 48-байтовый блок: X_0, X_1, X_2, . . ., X_{47}. Заполните первые 16 байтов X нулями, во вторые 16 байтов X скопируйте первые 16 байтов сообщения, а третьи 16 байтов X должны быть равны XOR первых и вторых 16 байтов X.

(4) Вот как выглядит функция сжатия:

$t = 0$

For $j = 0$ to 17

For $k = 0$ to 47

$t = X_t$ XOR S_t

$X_k = t$

$t = (t + j)$ mod 256

(5) Скопируйте во вторые 16 байтов X вторые 16 байтов сообщения, а третьи 16 байтов X должны быть равны XOR первых и вторых 16 байтов X. Выполните этап (4). Повторяйте этапы (5) и (4) по очереди для каждых 16 байтов сообщения.

(6) Выходом являются первые 16 байтов X.

Хотя в MD2 пока не было найдено слабых мест (см. [1262]), она работает медленнее большинства других предлагаемых хэш-функций.

18.7 Алгоритм безопасного хэширования (Secure Hash Algorithm, SHA)

NIST, вместе с NSA, для Стандарта цифровой подписи (Digital Signature Standard, см. Раздел 20.2) разработал Алгоритм безопасного хэширования (Secure Hash Algorithm, SHA) [1154 (Digital Signature Standard]. (Сам стандарт называется Стандарт безопасного хэширования (Secure Hash Standard, SHS), а SHA - это алгоритм, используемый в стандарте.) В соответствии с *Federal Register* [539]:

> Предлагается Федеральный стандарт обработки информации (Federal Information Processing Standard, FIPS) для Стандарта безопасного хэширования (Secure Hash Standard, SHS). Этот предложение определяет Алгоритм безопасного хэширования (Secure Hash Algorithm, SHA) для использования вместе со Стандартом цифровой подписи (Digital Signature Standard) . . ..

> Кроме того, для приложений, в которых не требуется цифровая подпись, SHA должен использоваться во всех Федеральных приложениях, в которых понадобится алгоритм безопасного хэширования.

И

> Этот Стандарт определяет Алгоритм безопасного хэширования (Secure Hash Algorithm, SHA), необходимый для обеспечения безопасности Алгоритма цифровой подписи (Digital Signature Algorithm, DSA). Для любого входного сообщения длиной меньше 2^{64} битов SHA выдает 160-битовый результат, называемый кратким содержанием сообщения. Далее, краткое содержание сообщения становится входом DSA, который вычисляет подпись для сообщения. Подписывание краткого содержания вместо всего сообщения часто повышает эффективность процесса, так как краткое содержание сообщения намного меньше, чем само сообщение. То же краткое содержание сообщения должно быть получено тем, кто проверяет подпись, если принятая им версия сообщения используется в качестве входа SHA. SHA называется безопасным, так как он разработан так, чтобы было вычислительно невозможно найти сообщение, соответствующее данному краткому содержанию сообщения или найти два различных сообщения с одинаковым кратким содержанием сообщения. Любые изменения, произошедшие при передаче сообщения, с очень высокой вероятностью приведут к изменению краткого содержания сообщения, и подпись не пройдет проверку. Принципы, лежащие в основе SHA, аналогичны использованным профессором Рональдом Л. Ривестом из MIT при проектировании алгоритма краткого содержания сообщения MD4 [1319]. SHA разработан по образцу упомянутого алгоритма.

SHA выдает 160-битовое хэш-значение, более длинное, чем у MD5.

Описание SHA

Во первых, сообщение дополняется, чтобы его длина была кратной 512 битам. Используется то же дополнение, что и в MD5: сначала добавляется 1, а затем нули так, чтобы длина полученного сообщения была на 64 бита меньше числа, кратного 512, а затем добавляется 64-битовое представление длины оригинального сообщения.

Инициализируются пять 32-битовых переменных (в MD5 используется четыре переменных, но рассматриваемый алгоритм должен выдавать 160-битовое хэш-значение):

A = 0x67452301

B = 0xefcdab89

C = 0x98badcfe

D = 0x10325476

E = 0xc3d2e1fO

Затем начинается главный цикл алгоритма. Он обрабатывает сообщение 512-битовыми блоками и продолжается, пока не исчерпаются все блоки сообщения.

Сначала пять переменных копируются в другие переменные: A в a, B в b, C в c, D в d и E в e.

Главный цикл состоит из четырех этапов по 20 операций в каждом (в MD5 четыре этапа по 16 операций в каждом). Каждая операция представляет собой нелинейную функцию над тремя из a, b, c, d и e, а затем выполняет сдвиг и сложение аналогично MD5. В SHA используется следующий набор нелинейных функций:

$f_t(X,Y,Z) = (X \wedge Y) \vee ((\neg X) \wedge Z)$, 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

в алгоритме используются следующие четыре константы:

K_t = 0x5a827999, для t=0 до 19

K_t = 0x6ed9eba1, для t=20 до 39

K_t = 0x8flbbcdc, для t=40 до 59

K_t = 0xca62c1d6, для t=60 до 79

(Если интересно, как получены эти числа, то: 0x5a827999 = $2^{1/2}/4$, 0x6ed9eba1 = $3^{1/2}/4$, 0x8flbbcdc = $5^{1/2}/4$, 0xca62c1d6 = $10^{1/2}/4$.)

Блок сообщения превращается из 16 32-битовых слов (M_0 по M_{15}) в 80 32-битовых слов (W_0 по W_{79}) с помощью следующего алгоритма:

$W_t = M_t$, для t = 0 по 15

$W_t = (W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16}) \lll 1$, для t = 16 по 79

(В качестве интересного замечания, в первоначальной спецификации SHA не было циклического сдвига вле-

во. Изменение "исправляет технический изъян, который делал стандарт менее безопасным, чем предполагалось " 1543]. NSA отказалось уточнить истинную причину изъяна.)

Если t - это номер операции (от 1 до 80), W_t представляет собой t-ый подблок расширенного сообщения, а $<<<s$ - это циклический сдвиг влево на s битов, то главный цикл выглядит следующим образом :

FOR t = 0 to 79

$TEMP = (a <<< 5) + f_t(b,c,d) + e + W_t + K_t$

$e = d$

$d = c$

$c = b <<< 30$

$b = a$

$a = TEMP$

На 11-й показана одна операция. Сдвиг переменных выполняет ту же функцию, которую в MD5 выполняет использование в различных местах различных переменных .

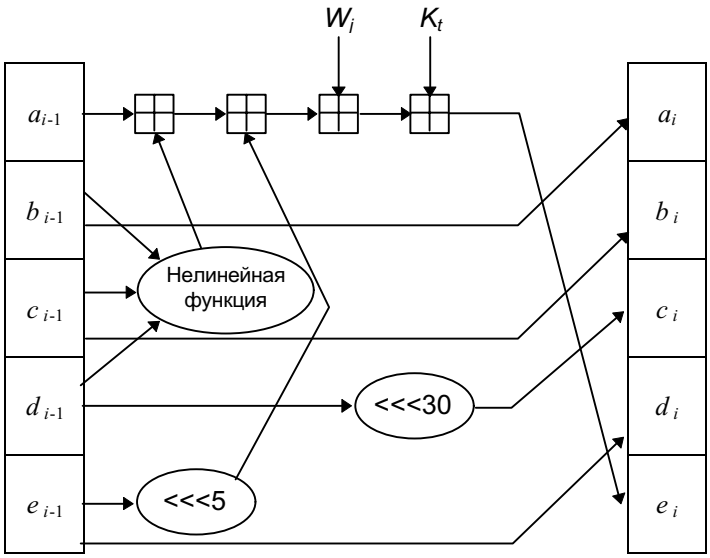


Рис. 18-7. Одна операция SHA.

После всего этого a, b, c, d и e добавляются к A, B, C D и E, соответственно, и алгоритм продолжается для следующего блока данных. Окончательным результатом служит объединение A, B, C D и E.

Безопасность SHA

SHA очень похожа на MD4, но выдает 160-битовое хэш-значение. Главным изменением является введение расширяющего преобразования и добавление выхода предыдущего шага в следующий с целью получения более быстрого лавинного эффекта. Рон Ривест опубликовал цели, преследуемые им при проектировании MD5, но разработчики SHA этого не сделали. Вот улучшения, внесенные Ривестом в MD5 относительно MD4, и их сравнение с SHA:

1. "Добавился четвертый этап." В SHA тоже. Однако в SHA на четвертом этапе используется та же функция f, что и на втором этапе.
2. "Теперь в каждом действии используется уникальная прибавляемая константа ." SHA придерживается схемы MD4, повторно используя константы для каждой группы их 20 этапов.
3. "Функция G на этапе 2 с $((X \wedge Y) \vee (X \wedge Z) \vee (Y \wedge Z))$ была изменена на $(X \wedge Z) \vee (Y \wedge (\neg Z))$, чтобы сделать G менее симметричной." В SHA используется версия функции из MD4: $(X \wedge Y) \vee (X \wedge Z) \vee (Y \wedge Z)$.
4. "Теперь каждое действие добавляется к результату предыдущего этапа . Это обеспечивает более быстрый лавинный эффект." Это изменение было внесено и в SHA. Отличие состоит в том, что в SHA добавлена пятая переменная к b, c и d, которые уже используются в f_t. Это незначительное изменение делает применения вскрытия MD5 ден Боером и Босселаерсом невозможным по отношению к SHA.
5. "Изменился порядок, в котором использовались подблоки сообщения на этапах 2 и 3 , чтобы сделать

шаблоны менее похожими." SHA в этом месте совершенно отличается, так как использует циклический код исправления ошибок.

6. "Значения циклического сдвига влево на каждом этапе были приближенно оптимизированы для ускорения лавинного эффекта. Четыре сдвига, используемые на каждом этапе, отличаются от значений, используемых на других этапах." SHA на каждом этапе использует постоянное значение сдвига. Это значение - взаимно простое число с размером слова, как и в MD4.

Это приводит к следующему заключению: SHA - это MD4 с добавлением расширяющего преобразования, дополнительного этапа и улучшенным лавинным эффектом. MD5 - это MD4 с улучшенным битовым хэшированием, дополнительным этапом и улучшенным лавинным эффектом.

Сведения об успешных криптографических вскрытиях SHA отсутствуют. Так как эта однонаправленная хэш-функция выдает 160-хэш-значение, она устойчивее к вскрытию грубой силой (включая вскрытие методом дня рождения), чем 128-битовые хэш-функции, рассматриваемые в этой главе.

18.8 RIPE-MD

RIPE-MD была разработана для проекта RIPE Европейского сообщества [1305] (см. раздел 25.7). Этот алгоритм представляет собой вариант MD4, разработанный так, чтобы противостоять известным методам криптографического вскрытия, и выдает 128-битовое хэш-значение. Внесены изменения в циклические сдвиги и порядок слов сообщения. Кроме того, параллельно работают две копии алгоритма, отличающиеся константами. После каждого блока результат обоих копий добавляется к переменным сцепления. По видимому, это повышает устойчивость алгоритма к криптоанализу.

18.9 HAVAL

HAVAL - это однонаправленная хэш-функция переменной длины [1646]. Она является модификацией MD5. HAVAL обрабатывает сообщение блоками по 1024 бита, в два раза большими, чем в MD5. Используется восемь 32-битовых переменных сцепления, в два раза больше, чем в MD5, и переменное число этапов, от трех до пяти (в каждом 16 действий). Функция может выдавать хэш-значения длиной 128, 160, 192, 224 или 256 битов.

HAVAL заменяет простые нелинейные функции MD5 на сильно нелинейные функции 7 переменных, каждая из которых удовлетворяет строгому лавинному критерию. На каждом этапе используется одна функция, но при каждом действии входные переменные переставляются различным образом. Используется новый порядок сообщения, и при каждом этапе (кроме первого этапа) используется своя прибавляемая константа. В алгоритме также используется два циклических сдвига.

Ядром алгоритма являются следующие действия:

$TEMP = (f(j,A,B,C,D,E,F,G) <<<7) + (H <<<11) + M[i][r(j)+K(j)]$

$H = G$; $G = F$; $F = E$; $E = D$; $D = C$; $C = B$; $B = A$; $A = TEMP$

Переменное количество этапов и переменная длина выдаваемого значения означают, что существует 15 версий алгоритма. Вскрытие MD5, выполненное ден Боером и Босселаерсом [203], неприменимо к HAVAL из-за циклического сдвига H.

18.10 Другие однонаправленные хэш-функции

MD3 является еще одной хэш-функцией, предложенной Роном Ривестом. Она имела ряд недостатков и никогда не выходила за пределы лаборатории, хотя ее описание недавно было опубликовано в [1335].

Группа исследователей из Университета Ватерлоо предложила однонаправленную хэш-функцию на базе итеративного возведения в степень в $GF(2^{593})$ [22]. По этой схеме сообщение разбивается на 593-битовые блоки. Начиная с первого блока блоки последовательно возводятся в степень. Показатель степени - это результат вычислений для предыдущего блока, первый показатель задается с помощью IV.

Айвэн Дамгард (Ivan Damgård) разработал однонаправленную хэш-функцию, основанную на проблеме рюкзака (см. раздел 19.2) [414], она может быть взломана примерно за 2^{32} операций [290, 1232, 787].

В качестве основы для однонаправленных хэш-функций предлагался и клеточный автомат Стива Вольфрама [1608]. Ранняя реализация [414] небезопасна [1052,404]. Другая однонаправленная хэш-функция, Cellhash [384, 404], и улучшенная версия, Subbash [384,402, 405], также основаны на клеточных автоматах и предназначены для аппаратной реализации. Boognish объединил принципы Cellhash и MD4 [402, 407]. StepRightUp также может быть реализована как хэш-функция [402].

Летом 1991 года Клаус Шнорр (Claus Schnorr) предложил однонаправленную хэш-функцию на базе дис-

кретного преобразования Фурье, названную FFT-Hash [1399]. Через несколько месяцев она была взломана двумя независимыми группами [403, 84]. Шнорр предложил новую версию, FFT-Hash II (предыдущая была переименована в FFT-Hash I) [1400], которая была взломана через несколько недель [1567]. Шнорр предложил дальнейшие модификации [1402, 1403] но, при данных обстоятельствах, они намного медленнее, чем другие алгоритмы этой главы. Еще одна хэш-функция, SL_2 [1526], небезопасна [315].

Дополнительную информацию по теории проектирования однонаправленных хэш-функций из однонаправленных функций и однонаправленных перестановок можно найти в [412, 1138, 1342].

18.11 Однонаправленные хэш-функции, использующие симметричные блочные алгоритмы

В качестве однонаправленных хэш-функций можно использовать симметричные блочные алгоритмы шифрования. Идея в том, что если безопасен блочный алгоритм, то и однонаправленная хэш-функция будет безопасной.

Самым очевидным способом является шифрование сообщения в режиме CBC или CFB с помощью фиксированного ключа и IV, хэш-значением будет последний блок шифротекста. Эти методы описаны в различных стандартах, использующих DES: оба режима в [1143], CBC в [1145], CFB в [55, 56, 54]. Этот способ не слишком подходит для однонаправленных хэш-функций, хотя он будет работать для MAC (см. раздел 18.14) [29].

Способ поумнее использует в качестве ключа блок сообщения, предыдущее хэш-значение в качестве входа, а текущее хэш-значение служит выходом.

Действительные хэш-функции даже еще сложнее. Размер блока обычно совпадает с длиной ключа, и размером хэш-значения будет длина блока. Так как большинство блочных алгоритмов 64-битовые, спроектирован ряд схем, дающих хэш-значение в два раза большее длины блока.

При условии, что хэш-функция правильна, безопасность этой схемы основана на безопасности используемой блочной функции. Однако есть и исключения. Дифференциальный криптоанализ лучше работает против блочных функций в хэш-функциях, чем против блочных функций, используемых для шифрования: ключ известен, поэтому можно использовать различные приемы. Для успеха нужна только одна правильная пара, и можно генерировать столько выбранного открытого текста, сколько нужно. Это направление освещается в [1263, 858, 1313].

Ниже приведен обзор различных хэш-функций, описанных в литературе [925, 1465, 1262]. Выводы о возможности вскрытия предполагают, что используемый блочный алгоритм безопасен, и лучшим вскрытием является вскрытие грубой силой.

Полезной мерой для хэш-функций, основанных на блочных шифрах, является **скорость хэширования**, или количество n-битовых блоков сообщения (n - это размер блока алгоритма), обрабатываемых при шифровании. Чем выше скорость хэширования, тем быстрее алгоритм. (Другое определение этого параметра дается в [1262], но определение, приведенное мной, более интуитивно и шире используется. Это может запутать.)

Схемы, в которых длина хэш-значения равна длине блока

Вот общая схема (см. 10-й):

$H_0 = I_H$, , где I_H - случайное начальное значение

$H_i = E_A(B) \oplus C$

где A, B и C могут быть либо M_i, H_{i-1}, ($M_i \oplus H_{i-1}$), либо константы (возможно равные 0). H_0 - это некоторое случайное начальное число I_H. Сообщение разбивается на части в соответствии с размером блока, M_i, обрабатываемые отдельно. Кроме того, используется вариант MD-усиления, возможно та же процедура дополнения, что и в MD5 и SHA.

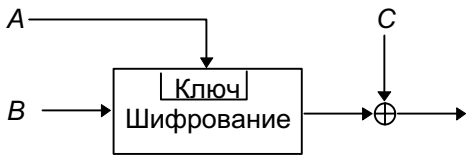


Рис. 18-8. Обобщенная хэш-функция, у которой длина хэш-значения равна длине блока

Табл. 18-1.
Безопасные хэш-функции, у которых

длина хэш-значения равна длине блока

$H_i = E_{H_{i-1}}(M_i) \oplus M_i$
$H_i = E_{H_{i-1}}(M_i \oplus H_{i-1}) \oplus M_i \oplus H_{i-1}$
$H_i = E_{H_{i-1}}(M_i) \oplus H_{i-1} \oplus M_i$
$H_i = E_{H_{i-1}}(M_i \oplus H_{i-1}) \oplus M_i$
$H_i = E_{M_i}(H_{i-1}) \oplus H_{i-1}$
$H_i = E_{M_i}(M_i \oplus H_{i-1}) \oplus M_i \oplus H_{i-1}$
$H_i = E_{M_i}(H_{i-1}) \oplus M_i \oplus H_{i-1}$
$H_i = E_{M_i}(M_i \oplus H_{i-1}) \oplus H_{i-1}$
$H_i = E_{M_i \oplus H_{i-1}}(M_i) \oplus M_i$
$H_i = E_{M_i \oplus H_{i-1}}(H_{i-1}) \oplus H_{i-1}$
$H_i = E_{M_i \oplus H_{i-1}}(M_i) \oplus H_{i-1}$
$H_i = E_{M_i \oplus H_{i-1}}(H_{i-1}) \oplus M_i$

Три различные переменные могут принимать одно из четырех возможных значений, поэтому всего сущес т-вует 64 варианта схем этого типа. Они все были изучены Бартом Пренелом (Bart Preneel) [1262].

Пятнадцать из них тривиально слабы, так как результат не зависит от одного из входов . Тридцать семь не-безопасны по более тонким причинам. В 17-й перечислены оставшиеся 12 безопасных схем: первые четыре безопасны против всех вскрытий (см. 9th), а последние 8 безопасны против всех типов вскрытий, кроме вскр ы-тия с фиксированной точкой, о котором в реальных условиях не стоит беспокоиться .

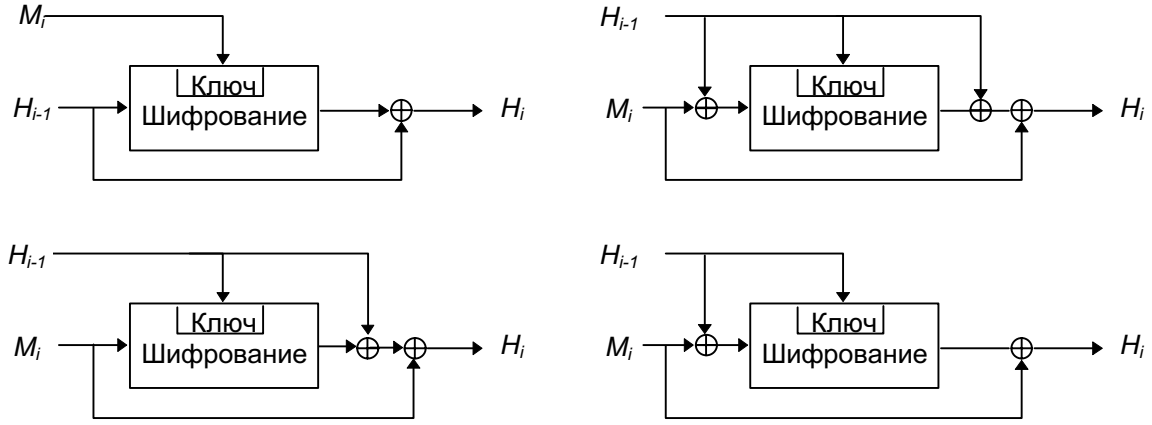


Рис. 18-9. Четыре безопасных хэш-функции, у которых длина хэш-значения равна длине блока

Первая схема была описана в [1028]. Третья схема была описана в [1555, 1105, 1106] и предлагалась в каче-стве стандарта ISO [766]. Пятая схема была предложена Карлом Майером (Carl Meyer), но в литературе обычно называется Davies-Meyer [1606, 1607, 434, 1028]. Десятая схема была предложена в качестве режима хэш-функции для LOKI [273].

Скорость хэширования первой, второй, третьей, четвертой, пятой и одиннадцатой схем равна 1 - длина кл ю-ча равна длине блока. Скорость хэширования других схем составляет k/n, где k -длина ключа. Это означает, что если длина ключа короче длины блока, то блок сообщения должен быть по длине равен ключу . Не рекомендует-ся, чтобы блок сообщения был длиннее ключа, даже если длина ключа алгоритма шифрования больше, чем длина блока.

Если блочный алгоритм подобно DES обладает свойством комплиментарности и слабыми ключами , для всех 12 схем существует возможность дополнительного вскрытия. Оно не слишком опасно и в действительности не стоит об это м беспокоиться. Однако вы можете обезопасить себя от такого вскрытия , зафиксировав значение второго и третьего битов ключа, равное "01" или "10" [1081,1107]. Конечно же это уменьшит длину k с 56 битов до 54 битов (для DES) и уменьшит скорость хэширования.

Было показано, что следующие схемы, описанные в литературе, небезопасны .

Эта схема [1282] была взломана в [369]:

$H_i = E_{M_i}(H_{i-1})$

Дэвис (Davies) и Прайс (Price) предложили вариант, в котором все сообщение циклически обрабатывается алгоритмом дважды [432, 433]. Вскрытие Копперсмита взламывает такую схему даже при небольшой вычисл и-тельной мощности [369]. В [1606] была показана небезопасность еще одной схемы [432, 458]:

$H_i = E_{M_i \oplus H_{i-1}}(H_{i-1})$

В [1028] была показана небезопасность следующей схемы (c - константа):

$H_i = E_c(M_i \oplus H_{i-1}) \oplus M_i \oplus H_{i-1}$

Модификация схемы Davies-Meyer

Лай (Lai) и Массей (Massey) модифицировали метод Davies-Meyer, чтобы можно было использовать шифр IDEA [930, 925]. IDEA использует 64-битовый блок и 128-битовый ключ. Вот предложенная ими схема:

$H_0 = I_H$, , где I_H - случайное начальное значение

$H_i = E_{H_{i-1}, M_i}(H_{i-1})$

Эта функция хэширует сообщение 64-битовыми блоками и выдает 64-битовое значение (см. 8-й).

Более простое вскрытие этой схемы, чем метод грубой силы, неизвестно.

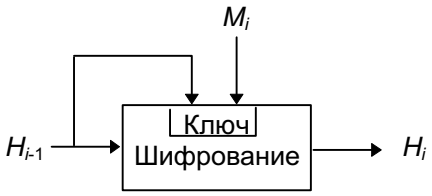


Рис. 18-10. Модификация схемы Davies-Meyer.

Preneel-Bosselaers-Govaerts-Vandewalle

Эта хэш-функция, впервые предложенная в [1266], выдает хэш-значение, в два раза большее длины блока алгоритма шифрования: при 64-битовом алгоритме получается 128-битовое хэш-значение.

При 64-битовом блочном алгоритме схема выдает два 64-битовых хэш-значения, G_i и H_i, объединение которых и дает 128-битовое хэш-значение. У большинства блочных алгоритмов длина блока равна 64 битам. Два соседних блока, L_i и R_i, размер каждого равен размеру блока, хэшируются вместе.

$G_0 = I_G$, где I_G - случайное начальное значение

$H_0 = I_H$, , где I_H - другое случайное начальное значение

$G_i = E_{L_i \oplus H_{i-1}}(R_i \oplus G_{i-1}) \oplus R_i \oplus G_{i-1} \oplus H_{i-1}$

$H_i = E_{L_i \oplus R_i}(H_{i-1} \oplus G_{i-1}) \oplus L_i \oplus G_{i-1} \oplus H_{i-1}$

Лай приводит вскрытие этой схемы, которое в некоторых случаях делает вскрытие методом дня рождения тривиальным [925, 926]. Пренел (Preneel) [1262] и Копперсмит (Coppersmith0 [372] также успешно взломали эту схему. Не используйте ее.

Quisquater-Girault

Эта схема, впервые предложенная в [1279], генерирует хэш-значение, в два раза большее длины блока. Ее скорость хэширования равна 1. Она использует два хэш-значения, G_i и H_i, и хэширует вместе два блока, L_i и R_i.

$G_0 = I_G$, где I_G - случайное начальное значение

$H_0 = I_H$, , где I_H - другое случайное начальное значение

$W_i = E_{L_i}(G_{i-1} \oplus R_i) \oplus R_i \oplus H_{i-1}$

$G_i = E_{R_i}(W_i \oplus L_i) \oplus G_{i-1} \oplus H_{i-1} \oplus L_i$

$H_i = W_i \oplus G_{i-1}$

Эта схема появилась в 1989 году в проекте стандарта ISO [764], но была заменена более поздней версией [765]. Проблемы безопасности этой схемы были описаны в [1107, 925, 1262, 372]. (В действительности, версия, описанная в материалах конференции, была после того, как версия, представленная на конференции, была вскрыта.) В ряде случаев сложность вскрытия методом дня рождения имеет равна 2^{39}, а не 2^{64}, как у вскрытия грубой. Не используйте эту схему.

LOKI с удвоенным блоком

Этот алгоритм представляет собой модификацию Quisquater-Cirault, специально спроектированную для ра-

боты с LOKI [273]. Все параметры - те же, что и в Quisquater-Girault.

$G_0 = I_G$, где I_G - случайное начальное значение

$H_0 = I_H$, , где I_H - другое случайное начальное значение

$W_i = E_{L_i \oplus G_{i-1}}(G_{i-1} \oplus R_i) \oplus R_i \oplus H_{i-1}$

$G_i = E_{R_i \oplus H_{i-1}}(W_i \oplus L_i) \oplus G_{i-1} \oplus H_{i-1} \oplus L_i$

$H_i = W_i \oplus G_{i-1}$

И снова в некоторых случаях вскрытие методом дня рождения оказывается тривиальным [925, 926, 1262, 372, 736]. Не используйте эту схему.

Параллельная схема Davies-Meyer

Это еще одна попытка создать алгоритм со скоростью хэширования 1, который выдает хэш-значение, в два раза большее длины блока. [736].

$G_0 = I_G$, где I_G - случайное начальное значение

$H_0 = I_H$, , где I_H - другое случайное начальное значение

$G_i = E_{L_i \oplus R_i}(G_{i-1} \oplus L_i) \oplus L_i \oplus H_{i-1}$

$H_i = E_{L_i}(H_{i-1} \oplus R_i) \oplus R_i \oplus H_{i-1}$

К сожалению эта схема тоже небезопасна [928, 861]. Оказывается, что хэш-функция удвоенной длины со скоростью хэширования, равной 1, не может быть безопаснее, чем Davies-Meyer [861].

Тандемная (Tandem) и одновременная (Abreast) схемы Davies-Meyer

Другой способ обойти ограничения, присущие блочным шифрам с 64-битовым ключом, использует алг о-ритм, подобный IDEA (см. раздел 13.9), с 64-битовым блоком и 128-битовым ключом. Следующие две схемы выдают 128-битовхэш-значение, а их скорость хэширования равна $^1/_2$ [930, 925].

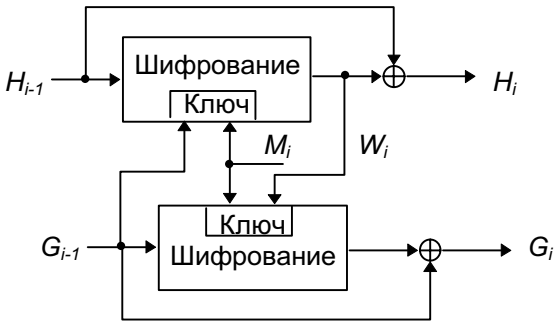


Рис. 18-11. Тандемная (Tandem) схема Davies-Meyer.

В первой схеме две модифицированные функции Davies-Meyer работают тандемом, конвейерно (см. 7-й).

$G_0 = I_G$, где I_G - случайное начальное значение

$H_0 = I_H$, , где I_H - другое случайное начальное значение

$W_i = E_{G_{i-1}, M_i}(H_{i-1})$

$G_i = G_{i-1} \oplus E_{M_i, W_i}(G_{i-1})$

$H_i = W_i \oplus H_{i-1}$

В следующей схеме используются две модифицированные функции, работающие одновременно (см. 6-й).

$G_0 = I_G$, где I_G - случайное начальное значение

$H_0 = I_H$, , где I_H - другое случайное начальное значение

$G_i = G_{i-1} \oplus E_{M_i, H_{i-1}}(\neg G_{i-1})$

$H_i = H_{i-1} \oplus E_{G_{i-1}, M_i}(H_{i-1})$

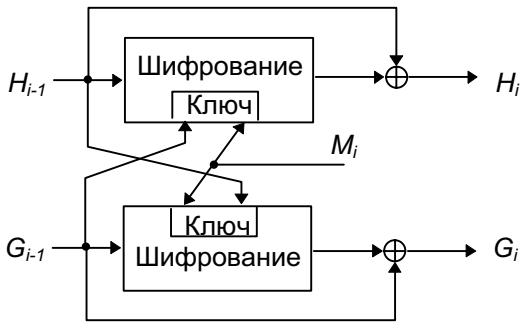


Рис. 18-12. Одновременная (Abreast) схема Davies-Meyer.

В обеих схемах два 64-битовых значения, G_i и H_i, объединяются, образуя единое 128-битовое хэш-значение.

Насколько известно, безопасность 128-битовой хэш-функции этих алгоритмов идеальна: для обнаружения сообщения с заданным хэш-значением требуется 2^{128} попыток, а для нахождения двух случайных сообщений с одинаковым хэш-значением - 2^{64} попыток, при условии, что лучшим способом вскрытия является применение грубой силы.

MDC-2 и MDC-4

MDC-2 и MDC-4 разработаны в IBM [1081, 1079]. В настоящее время изучается вопрос использования MDC-2, иногда называемой Meyer-Schilling, в качестве стандарта ANSI и ISO [61, 765], этот вариант был предложен в [762]. MDC-4 определена для проекта RIPE [1305] (см. раздел 25.7). Спецификация использует DES в качестве блочной функции, хотя теоретически может быть использован любой блочный алгоритм.

Скорость хэширования MDC-2 равна $^1/_2$, длина хэш-значения этой функции в два раза больше размера блока.Ее схема показана на 5-й. MDC-4 также выдает хэш-значение в два раза большее размера блока, а ее ск о-рость хэширования равна $^1/_4$ (см. 4-й).

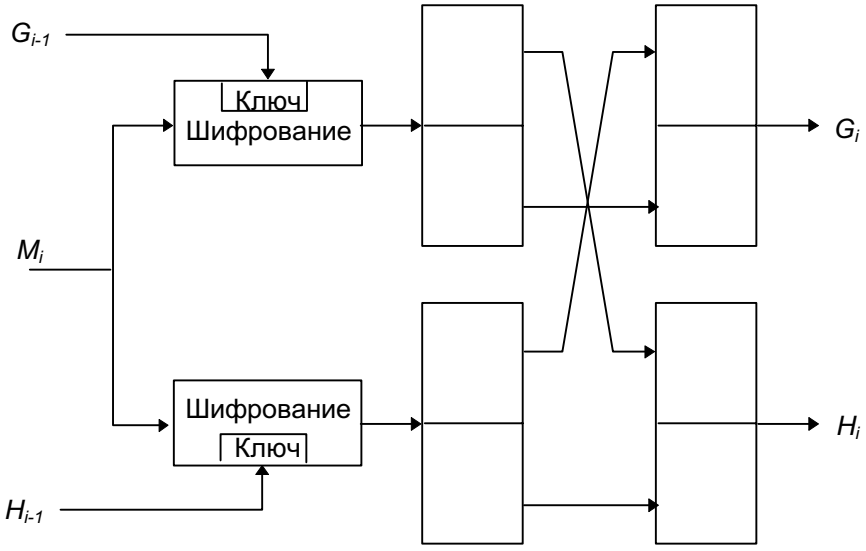


Рис. 18-13. MDC-2.

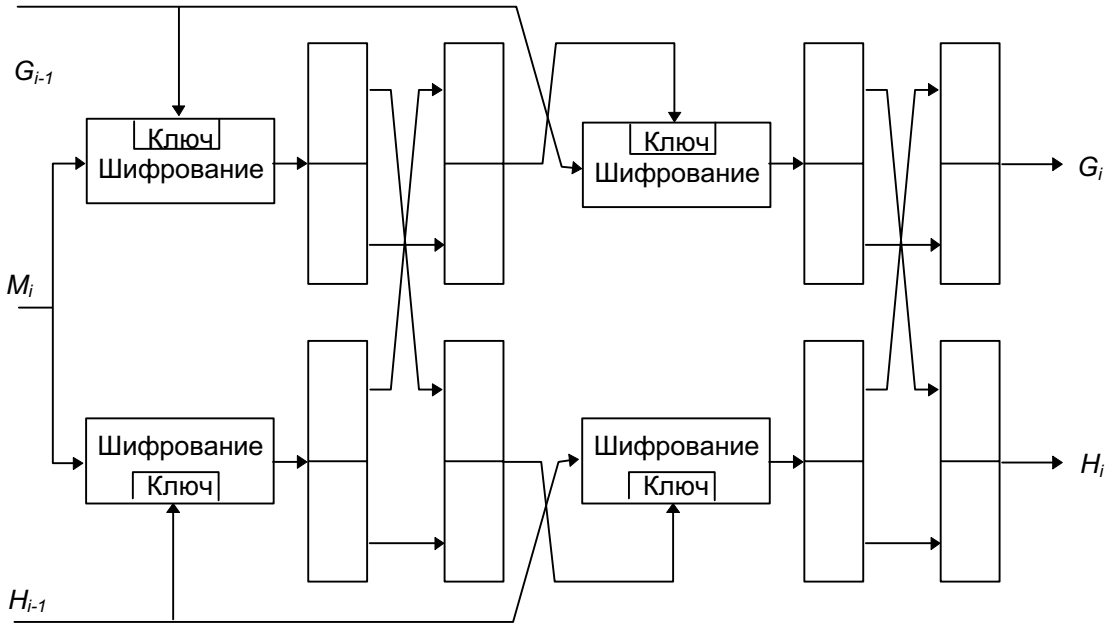


Рис. 18-14. MDC-4.

Эти схемы были проанализированы в [925, 1262]. Они безопасны с учетом сегодняшних возможностей вычислительной техники, но их надежность не так велика, как хотелось разработчикам. Их устойчивость к дифференциальному криптоанализу при DES в качестве блочного алгоритма была рассмотрена в [1262].

MDC-2 и MDC-4 запатентованы [223].

Хэш-функция AR

Хэш-функция AR была разработана Algorithmic Research, Ltd. и затем распространена ISO только для информации [767]. Ее базовая структура является вариантом используемого блочного шифра (DES в упомянутой статье) в режиме CBC. Выполняется XOR последних двух блоков шифротекста, константы и текущего блока сообщения, результат шифруется алгоритмом. Хэш-значением являются последние вычисленные два блока шифротекста. Сообщение обрабатывается дважды, двумя различными ключами, поэтому скорость хэширования равна $^1/_2$. Первым ключом служит 0x0000000000000000, вторым - 0x2a41522f4446502a, а значение константы c равно 0x0123456789abcdef. Результат сжимается до одного 128-битового хэш-значения. Подробности приведены в [750].

$H_i = E_K(M_i \oplus H_{i-1} \oplus H_{i-2} \oplus c) \oplus M_i$

Функция выглядит привлекательной, но не является безопасной. После некоторой значительной предобработки становится возможным легко находить сообщения с одинаковым хэш-значением [416].

Хэш-функция ГОСТ

Эта хэш-функция появилась в России и определена в стандарте ГОСТ Р 34.11.94 [657]. В ней используется блочный алгоритм ГОСТ (см. раздел 14.1), хотя теоретически может использоваться любой блочный алгоритм с 64-битовым блоком и 256-битовым ключом. Функция выдает 256-битовое хэш-значение.

Функция сжатия, $H_i = f(M_i,H_{i-1})$ (оба операнда - 256-битовые величины) определяется следующим образом:

(1) При помощи линейного смешивания M_i, H_{i-1} и некоторых констант генерируется четыре ключа шифрования ГОСТ.

(2) Каждый ключ используется для шифрования отличных 64 битов H_{i-1} в режиме ECB. Полученные 256 битов сохраняются во временной переменной S.

(3) H_i является сложной, хотя и линейной функцией S, M_i и H_{i-1}.

Хэш-значение последнего блока сообщения не является его окончательным хэш-значением. На деле используется три переменные сцепления: H_n - это хэш-значение последнего блока, Z - это XOR всех блоков сообщения, а L - длина сообщения. С использованием этих переменных и дополненного последнего блока M', окончательное хэш-значение равно:

$H = f(Z \oplus M', f(L, f(M', H_n)))$

Документация немного запутана (и на русском языке), но я думаю, что понял все правильно . Во всяком случае эта хэш-функция определена как часть российского Стандарта цифровой подписи (см. раздел 20.3).

Другие схемы

Ральф Меркл предложил схему, использующую DES, но она медленна - обрабатывает только семь битов сообщения за итерацию, и каждая итерация состоит из двух шифрований DES [1065, 1069]. Другая схема [1642, 1645] небезопасна [1267], когда-то она предлагалась в качестве стандарта ISO.

18.12 Использование алгоритмов с открытым ключом

В качестве однонаправленной хэш-функции можно использовать и алгоритм шифрования с открытым ключом в режиме сцепления блоков. Если затем выбросить личный ключ, то взломать хэш-функцию будет также трудно, как и прочитать сообщение без личного ключа.

Вот пример, использующий RSA. Если M - это хэшируемое сообщение, n - произведение двух простых чисел p и q, а e - другое большое число, взаимно простое с $(p - 1)(q - 1)$, то хэш-функция, $H(M)$, будет равна

$$H(M) = M^e \bmod n$$

Еще проще использовать одно сильное простое число в качестве модуля p. Тогда:

$$H(M) = M^e \bmod p$$

Вскрытие этой проблемы возможно не легче, чем поиск дискретного логарифма e. Проблема этого алгоритма состоит в том, что он намного медленнее, чем другие обсуждаемые алгоритмы . По этой причине я не советую его.

18.13 Выбор однонаправленной хэш-функции

Лучшими кажутся SHA, MD5 и схемы, основанные на блочных шифрах. Другие на самом деле не были исследованы в достаточной степени. Я голосую за SHA. У нее более длинное хэш-значение, чем у MD5, она быстрее, чем многие схемы с блочными шифрами, и разработана NSA. Я верю в криптоаналитические возможности NSA, даже если они не публикуют свои результаты.

В 16-й для сравнения приведены временные соотношения для некоторых хэш-функций . They are meant for comparison purposes only.

Табл. 18-2.
Скорости шифрования некоторых хэш-функций на i486SX/33 МГц

Алгоритм	Длина хэш-значения	Скорость шифрования (Кбайт/с)
Одновременная схема Davies-Meyer (с IDEA)	128	22
Davies-Meyer (с DES)	64	9
Хэш-функция ГОСТ	256	11
HAVAL (3 прохода)	переменная	168
HAVAL (4 прохода)	переменная	118
HAVAL (5 прохода)	переменная	95
MD2	128	23
MD4	128	236
MD5	128	174
N-хэш (12 этапов)	128	29
N-хэш (15 этапов)	128	24
RIPE-MD	128	182
SHA	160	75
Snerfu (4 прохода)	128	48
Snerfu (8 проходов)	128	23

18.14 Коды проверки подлинности сообщения

Код проверки подлинности сообщения (message authentication code, MAC) - это зависящая от ключа однонаправленная хэш-функция. Коды MAC обладают теми же свойствами, что и рассмотренные ранее хэш-функции, но они, кроме того, включают ключ. (Это не означает, что вы можете опубликовать ключ MAC и использовать MAC как однонаправленную хэш-функцию.) Только владелец идентичного ключа может проверить хэш-значение. Коды MAC очень полезны для обеспечения проверки подлинности без нарушения безопасности.

Коды MAC могут быть использованы для проверки подлинности файлов, которыми обмениваются пользователи. Также они могут быть использованы одним пользователем для проверки, не изменились ли его файлы, может быть из-за вируса. Пользователь может вычислить MAC его файлов и сохранить эти значения в таблице. Если пользователь воспользуется вместо MAC однонаправленной хэш-функцией, то вирус может вычислить новые хэш-значения после заражения файлов и заменить элементы таблицы. С MAC вирус не сможет этого добиться, так как ключ вирусу неизвестен.

Простым способом преобразовать однонаправленную хэш-функцию в MAC является шифрование хэш-значения симметричным алгоритмом. Любой MAC может быть преобразован в однонаправленную хэш-функцию с помощью раскрытия ключа.

CBC-MAC

Простейший способ создать зависящую от ключа однонаправленную хэш-функцию - шифрование сообщения блочным алгоритмом в режимах CBC или CFB. Хэш-значением является последний шифрованный блок, зашифрованный в режимах CBC или CFB. Метод CBC определен в ANSI X9.9 [54], ANSI X9.19 [56], ISO 8731-1 [759], ISO 9797 [763] и австралийском стандарте [1496]. Дифференциальный криптоанализ может вскрыть эту схему, если в качестве блочного алгоритма используется DES с уменьшенным числом этапов или FEAL [1197].

Потенциальная проблема, связанная с безопасностью этого метода, состоит в том, что получатель должен знать ключ, и этот ключ позволяет ему генерировать сообщения с тем же хэш-значением, что и у присланного сообщения, с помощью дешифрирования в обратном направлении.

Алгоритм проверки подлинности сообщения (Message Authenticator Algorithm, MAA)

Этот алгоритм является стандартом ISO [760]. Он выдает 32-битовое хэш-значение и был спроектирован для мэйнфреймов с быстрыми инструкциями умножения [428].

$v = v <<< 1$

$e = v \oplus w$

$x = ((((e + y) \bmod 2^{32}) \vee A \wedge C) * (x \oplus M_i)) \bmod 2^{32}\text{-}1$

$y = ((((e + x) \bmod 2^{32}) \vee B \wedge D) * (y \oplus M_i)) \bmod 2^{32}\text{-}1$

Эти действия повторяются для каждого блока сообщения, M_i, и результирующее хэш-значение получается с помощью XOR x и y. Переменные v и e зависят от ключа. A, B, C и D являются константами.

Возможно, этот алгоритм широко используется, но я не верю, что он достаточно безопасен. Он был разработан давным давно и не слишком сложен.

Двунаправленный MAC

Этот MAC выдает хэш-значение, которое в два раза длиннее блока алгоритма [978). Сначала для сообщения вычисляется CBC-MAC. Затем вычисляется CBC-MAC сообщения с обратным порядком блоков. Двунаправленный MAC просто является объединением этих двух значений. К сожалению эта схема небезопасна [1097].

Методы Джунемана

Этот MAC также называют квадратичным конгруэнтным кодом обнаружения манипуляции (quadratic congruential manipulation detection code, QCMDC) [792, 789]. Сначала разделим сообщение на m-битовые блоки. Затем:

$H_0 = I_H$, , где I_H - секретный ключ

$H_i = (H_{i-1} + M_i)^2 \bmod p$, где p - простое число, меньшее 2^m-1, а + обозначает целочисленное сложение.

Джунеман (Jueneman) предлагает $n = 16$ и $p = 2^{31}$-1. В [792] он также предлагает, чтобы H_1 использовался в качестве дополнительного ключа, а действительное сообщение начиналось бы с H_2.

Из-за множества вскрытий типа дня рождения, выполненных в сотрудничестве с Доном Копперсмитом, Джунеман предложил вычислять QCMDC четыре раза, используя результат одной итерации в качестве IV для

следующей итерации, а затем результаты объединяются в 128-битовое хэш-значение [793]. В дальнейшем эта идея была усилена за счет параллельного выполнения четырех итераций с поперечными связями между ними [790, 791]. Эта схема была взломана Копперсмитом [376].

В другом варианте [432, 434] операция сложения заменена XOR, и используются блоки сообщения, намного меньшие p. Кроме того, был задан H_0, что превратило алгоритм в однонаправленную хэш-функцию без ключа. После того, как эта схема была вскрыта [612], она была усилена для использования в качестве части проекта European Open Shop Information-TeleTrust [1221], процитирована в CCITT X.509 [304] и принята ISO в 10118 [764, 765]. К сожалению Копперсмит взломал и эту схему [376]. В ряде исследований изучалась возможность использовать отличные от 2 основания экспоненты [603], но ни одно не оказалось перспективным.

RIPE-MAC

RIPE-MAC был изобретен Бартом Пренелом [1262] и использован в проекте RIPE [1305] (см. раздел 18.8). Он основан на ISO 9797 [763] и использует DES в качестве функции блочного шифрования. Существует два варианта RIPE-MAC: один, который использует обычный DES, называется RIPE-MAC1, а другой, использующий для еще большей безопасности тройной DES, называется RIPE-MAC3. RIPE-MAGI использует одно шифрование DES на 64-битовый блок сообщения, а RIPE-MAC3 - три.

Алгоритм состоит из трех частей. Во первых, сообщение увеличивается так, чтобы его длина была кратна 64 битам. Затем, увеличенное сообщение разбивается на 64-битовые блоки. Для хэширования этих блоков в один блок используется функция сжатия, зависящая от секретного ключа. На этом этапе используется либо DES, либо тройной DES. Наконец, выход этой функции сжатия подвергается еще одному DES-шифрованию с другим ключом, полученным из ключа, используемого при сжатии. Подробности можно найти в [1305].

IBC-хэш

IBC-хэш - это еще один MAC, используемый в проекте RIPE [1305] (см. раздел 18.8). Он интересен потому, что его безопасность доказана, вероятность успешного вскрытия может быть оценена количественно. К сожалению каждое сообщение должно хэшироваться новым ключом. Выбранный уровень безопасности ограничивает максимальный размер хэшируемого сообщения, чего не делает ни одна другая из рассмотренных в этой главе функция. С учетом этих соображений в отчете RIPE рекомендуется, чтобы IBC-хэш использовалась бы только для длинных, редко посылаемых сообщений. Ядром функции является

$h_i = ((M_i \bmod p) + v) \bmod 2^n$

Секретный ключ представляет собой пару p и v, где p - n-битовое простое число, а v - случайное число, меньшее 2^n. Значения M_i получаются с помощью строго определенной процедуры дополнения. Вероятности вскрыть как однонаправленность, так и устойчивость к столкновениям, могут быть оценены количественно, и пользователи, меняя параметры, могут выбрать нужный уровень безопасности.

Однонаправленная хэш-функция MAC

В качестве MAC может быть использована и однонаправленная хэш-функция [1537]. Пусть Алиса и Боб используют общий ключ K, и Алиса хочет отправить Бобу MAC сообщения M. Алиса объединяет K и M, и вычисляет однонаправленную хэш-функцию объединения: $H(K,M)$. Это хэш-значение и является кодом MAC. Так как Боб знает K, он может воспроизвести результат Алисы, а Мэллори, которому ключ неизвестен, не сможет это сделать.

Со методами MD-усиления этот способ работает, но есть серьезные проблемы. Мэллори всегда может добавить новые блоки к концу сообщения и вычислить правильный MAC. Это вскрытие может быть предотвращено, если к началу сообщения добавить его длину, но Пренел сомневается в этой схеме [1265]. Лучше добавлять ключ к концу сообщения, $H(M,K)$, но при этом также возникают проблемы [1265]. Если H однонаправленная функция, которая не защищена от столкновений, Мэллори может подделывать сообщения. Еще лучше $H(K,M,K)$ или $H(K_1,M,K_2)$, где K_1 и K_2 различны [1537]. Пренел не уверен и в этом [1265].

Безопасными кажутся следующие конструкции:

$H(K_1, H(K_2, M))$

$H(K, H(K,M))$

$H(K, p,M,K))$, где p дополняет K до полного блока сообщения.

Лучшим подходом является объединение с каждым блоком сообщения по крайней мере 64 битов ключа. Это делает однонаправленную функцию менее эффективной, так как уменьшаются блоки сообщения, но так она становится намного безопаснее [1265].

Или используйте однонаправленную хэш-функцию и симметричный алгоритм. Сначала хэшируйте файл,

потом зашифруйте хэш-значение. Это безопаснее, чем сначала шифровать файл, а затем хэшировать зашифр о-ванный файл, но эта схема чувствительна к тому же вскрытию, что и конструкция $H(M,K)$ [1265].

MAC с использованием потокового шифра

Эта схема MAC использует потоковые шифры (см. 3-й) [932]. Криптографически безопасный генератор псевдослучайных битов демультиплексирует поток сообщения на два подпотока. Если на выходе генератора битов k_i единица, то текущий бит сообщения m_i отправляется в первый подпоток, если ноль, то m_i отправляется во второй подпоток. Каждый подпоток отправляется на свой LFSR (раздел 16.2). Выходом MAC просто является конечное состояние обоих регистров.

К несчастью этот метод небезопасен по отношению к небольшим изменениям в сообщении [1523]. Например, если изменить последний бит сообщения, то для создания поддельного MAC нужно будет изменить только 2 бита соответствующего MAC; это может быть выполнено с заметной вероятностью. Автор предлагает более безопасный, и более сложный, вариант.

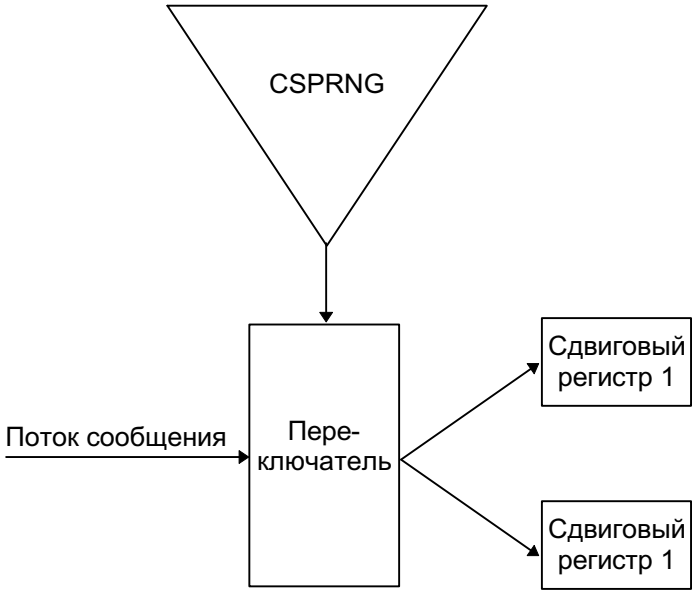


Рис. 18-15. MAC с использованием потокового шифра

Глава 19 Алгоритмы с открытыми ключами

19.1 Основы

Концепция криптографии с открытыми ключами была выдвинута Уитфилдом Диффи (Whitfield Diffie) и Мартином Хеллманом (Martin Hellman), и независимо Ральфом Мерклом (Ralph Merkle). Их вкладом в криптографию было убеждение, что ключи можно использовать парами - ключ шифрования и ключ дешифрирования - и что может быть невозможно получить один ключ из другого (см. Раздел 2.5). Диффи и Хеллман впервые представили эту идею на Национальной компьютерной конференции (National Computer Conference) 1976 года [495], через несколько месяцев была опубликована их основополагающая работа "New Directions in Cryptography" ("Новые направления в криптографии") [496]. (Из-за бесстрастного процесса публикации первый вклад Меркла в эту область вышел появился только в 1978 году [1064].)

С 1976 года было предложено множество криптографических алгоритмов с открытыми ключами . Многие из них небезопасны. Из тех, которые являются безопасными, многие непригодны для практической реализации . Либо они используют слишком большой ключ, либо размер полученного шифротекста намного превышает размер открытого текста.

Немногие алгоритмы являются и безопасными, и практичными . Обычно эти алгоритмы основаны на одной из трудных проблем, рассмотренных в разделе 11.2. Некоторые из этих безопасных и практичных алгоритмов подходят только для распределения ключей. Другие подходят для шифрования (и для распределения ключей). Третьи полезны только для цифровых подписей. Только три алгоритма хорошо работают как при шифровании, так и для цифровой подписи: RSA, ElGamal и Rabin. Все эти алгоритмы медленны. Они шифруют и дешифруют данные намного медленнее, чем симметричные алгоритмы. Обычно их скорость недостаточна для шифрования больших объемов данных.

Гибридные криптосистемы (см. раздел 2.5) позволяют ускорить события: для шифрования сообщения используется симметричный алгоритм со случайным ключом, а алгоритм с открытым ключом применяется для шифрования случайного сеансового ключа.

Безопасность алгоритмов с открытыми ключами

Так как у криптоаналитика есть доступ к открытому ключу, он всегда может выбрать для шифрования любое сообщение. Это означает, что криптоаналитик при заданном $C = E_K(P)$ может попробовать угадать значение P и легко проверить свою догадку. Это является серьезной проблемой, если количество возможных открытых текстов настолько мало, что делает возможным исчерпывающий поиск, но эту проблему легко можно решить, дополняя сообщения строкой случайных битов. Это приводит к тому, что идентичным открытым текстам соответствуют различные шифротексты. (Более подробно эта идея описана в разделе 23.15.)

Это особенно важно, если алгоритм с открытым ключом используется для шифрования сеансового ключа . Ева может создать базу данных всех возможных сеансовых ключей, зашифрованных открытым ключом Боба . Конечно, это потребует много времени и памяти, но взлом грубой силой разрешенного к экспорту 40-битового ключа или 56-битового ключа DES потребует намного больше времени и памяти. Как только Ева создаст такую базу данных, она получит ключ Боба и сможет читать его почту .

Алгоритмы с открытыми ключами спроектированы так, чтобы противостоять вскрытиям с выбранным открытым текстом. Их безопасность основана как на трудности получения секретного ключа по открытому, так и на трудности получить открытый текст по шифротексту. Однако большинство алгоритмов с открытым ключом особенно чувствительны к вскрытию с выбранным шифротекстом (см. раздел 1.1).

В системах, в которых операция, обратная шифрованию, используется для цифровой подписи, это вскрытие невозможно предотвратить, если для шифрования и подписей использовать одинаковые ключи .

Следовательно, важно увидеть всю систему целиком, а не только составные части . Хорошие протоколы с открытыми ключами спроектированы таким образом, чтобы различные стороны не могли расшифровать произвольные сообщения, генерированные другими сторонами, - хорошим примером являются протоколы доказательства идентичности (см. раздел 5.2).

19.2 Алгоритмы рюкзака

Первым алгоритмом для обобщенного шифрования с открытым ключом стал алгоритм рюкзака, разработанный Ральфом Мерклом и Мартином Хеллманом [713, 1074]. Он мог быть использован только для шифрования, хотя позднее Ади Шамир адаптировал систему для цифровой подписи [1413]. Безопасность алгоритмов рюкзака опирается на проблему рюкзака, **NP-полную** проблему. Хотя позже было обнаружено, что этот алгоритм небезопасен, его стоит изучить, так как он демонстрирует возможность применения **NP-полной** проблемы

в криптографии с открытыми ключами.

Проблема рюкзака несложна. Дана куча предметов различной массы, можно ли положить некоторые из этих предметов в рюкзак так, чтобы масса рюкзака стала равна определенному значению ? Более формально, дан набор значений M_1, M_2, . . . , M_n и сумма S, вычислить значения b_i, такие что

$S = b_1M_1 + b_2M_2 + . . . + b_nM_n$

b_i может быть либо нулем, либо единицей. Единица показывает, что предмет кладут в рюкзак, а ноль - что не кладут.

Например, массы предметов могут иметь значения 1, 5, 6, 11, 14 и 20. Вы можете упаковать рюкзак так, чтобы его масса стала равна 22, использовав массы 5, 6 и 11. Невозможно упаковать рюкзак так, чтобы его масса была равна 24. В общем случае время, необходимое для решения этой проблемы, с ростом количества предметов в куче растет экспоненциально.

В основе алгоритма рюкзака Меркла-Хеллмана лежит идея шифровать сообщение как решение набора проблем рюкзака. Предметы из кучи выбираются с помощью блока открытого текста, по длине равного количеству предметов в куче (биты открытого текста соответствуют значениям b), а шифротекст является полученной суммой. Пример шифротекста, зашифрованного с помощью проблемы рюкзака, показан на .

Открытый текст	1 1 1 0 0 1	0 1 0 1 1 0	0 0 0 0 0 0	0 1 1 0 0 0
Рюкзак	1 5 6 11 14 20	1 5 6 11 14 20	1 5 6 11 14 20	1 5 6 11 14 20
Шифротекст	1+5+6+20=32	5+11+14=30	0=0	5+6=11

Рис. 19-1. Шифрование с рюкзаками

Фокус в том, что на самом деле существуют две различные проблемы рюкзака , одна решается за линейное время, а другая, как считается, - нет. Легкую проблему можно превратить в трудную. Открытый ключ представляет собой трудную проблему, которую легко использовать для шифрования, но невозможно для дешифрирования сообщений. Закрытый ключ является легкой проблемой, давая простой способ дешифрировать сообщения . Тому, кто не знает закрытый ключ, придется попытаться решить трудную проблему рюкзака .

Сверхвозрастающие рюкзаки

Что такое легкая проблема рюкзака? Если перечень масс представляет собой **сверхвозрастающую последовательность**, то полученную проблему рюкзака легко решить. Сверхвозрастающая последовательность - это последовательность, в которой каждой член больше суммы всех предыдущих членов . Например, последовательность {1,3,6,13,27,52} является сверхвозрастающей, а {1,3,4,9, 15,25} - нет.

Решение **сверхвозрастающего рюкзака** найти легко. Возьмите полный вес и сравните его с самым большим числом последовательности. Если полный вес меньше, чем это число, то его не кладут в рюкзак . Если полный вес больше или равен этому числу, то оно кладется в рюкзак . Уменьшим массу рюкзака на это значение и перейдем к следующему по величине числу последовательности. Будем повторять, пока процесс не закончится. Если полный вес уменьшится до нуля, то решение найдено. В противном случае , there isn't.

Например, пусть полный вес рюкзака - 70, а последовательность весов {2,3,6, 13,27,52}. Самый большой вес, 52, меньше 70, поэтому кладем 52 в рюкзак. Вычитая 52 из 70, получаем 18. Следующий вес, 27, больше 18, поэтому 27 в рюкзак не кладется. вес, 13,меньше 18, поэтому кладем 13 в рюкзак. Вычитая 13 из 18, получаем 5. Следующий вес, 6, больше 5, поэтому 6 не кладется в рюкзак. Продолжение этого процесса покажет, что и 2, и 3 кладутся в рюкзак, и полный вес уменьшается до 0, что сообщает о найденном решении. Если бы это был блок шифрования методом рюкзака Меркла-Хеллмана , открытый текст, полученный из значения шифротекста 70, был бы равен 110101.

Не сверхвозрастающие, или нормальные, рюкзаки представляют собой трудную проблему - быстрого алгоритма для них не найдено. Единственным известным способом определить, какие предметы кладутся в рюкзак, является методическая проверка возможных решений, пока вы не наткнетесь на правильное . Самый быстрый алгоритм, принимая во внимание различную эвритсику , имеет экспоненциальную зависимость от числа возможных предметов. Добавьте к последовательности весов еще один член, и найти решение станет вдвое труднее. Это намного труднее сверхвозрастающего рюкзака, где, если вы добавите один предмет к последовательности, поиск решения увеличится на одну операцию .

Алгоритм Меркла-Хеллмана основан на этом свойстве. Закрытый ключ является последовательностью весов проблемы сверхвозрастающего рюкзака. Открытый ключ - это последовательность весов проблемы нормального рюкзака с тем же решением. Меркл и Хеллман, используя модульную арифметику, разработали способ преобразования проблемы сверхвозрастающего рюкзака в проблему нормального рюкзака.

Создание открытого ключа из закрытого

Рассмотрим работу алгоритма, не углубляясь в теорию чисел: чтобы получить нормальную последовательность рюкзака, возьмем сверхвозрастающую последовательность рюкзака, например, {2,3,6,13,27,52}, и умножим по модулю *m* все значения на число *n*. Значение модуля должно быть больше суммы всех чисел последовательности, например, 105. Множитель должен быть взаимно простым числом с модулем, например, 31. Нормальной последовательностью рюкзака будет

2*31 mod 105 = 62

3*31 mod 105 = 93

6*31 mod 105 = 81

13*31 mod 105 = 88

27*31 mod 105 = 102

52*31 mod 105 = 37

Итого - {62,93,81,88,102,37}.

Сверхвозрастающая последовательность рюкзака является закрытым ключом, а нормальная последовательность рюкзака - открытым.

Шифрование

Для шифрования сообщение сначала разбивается на блоки, равные по длине числу элементов последовательности рюкзака. Затем, считая, что единица указывает на присутствие члена последовательности, а ноль - на его отсутствие, вычисляем полные веса рюкзаков - по одному для каждого блока сообщения.

Например, если сообщение в бинарном виде выглядит как 011000110101101110, шифрование, использующее предыдущую последовательность рюкзака, будет происходить следующим образом:

сообщение = 011000 110101 101110

011000 соответствует 93 + 81 = 174

110101 соответствует 62 + 93 + 88 + 37 = 280

101110 соответствует 62 + 81 + 88 + 102 = 333

Шифротекстом будет последовательность 174,280,333

Дешифрирование

Законный получатель данного сообщения знает закрытый ключ: оригинальную сверхвозрастающую последовательность, а также значения *n* и *m*, использованные для превращения ее в нормальную последовательность рюкзака. Для дешифрирования сообщения получатель должен сначала определить n^{-1}, такое что $n(n^{-1}) \equiv 1 \pmod{m}$. Каждое значение шифротекста умножается на $n^{-1} \bmod m$, а затем разделяется с помощью закрытого ключа, чтобы получить значения открытого текста.

В нашем примере сверхвозрастающая последовательность - {2,3,6,13,27,52), *m* равно 105, а *n* - 31. Шифротекстом служит 174,280,333. В этом случае n^{-1} равно 61, поэтому значения шифротекста должны быть умножены на 61 mod 105.

174*61 mod 105 = 9 = 3 + 6, что соответствует 011000

280*61 mod 105 = 70 = 2 + 3 + 13 + 52, что соответствует 110101

333*61 mod 105 = 48 = 2 + 6 + 13 + 27, что соответствует 101110

Расшифрованным открытым текстом является 011000 110101 101110.

Практические реализации

Для последовательности из шести элементов нетрудно решить задачу рюкзака, даже если последовательность не является сверхвозрастающей. Реальные рюкзаки должны содержать не менее 250 элементов. Длина каждого члена сверхвозрастающей последовательности должна быть где-то между 200 и 400 битами, а длина модуля должна быть от 100 до 200 битов. Для получения этих значений практические реализации используют генераторы случайной последовательности.

Вскрывать подобные рюкзаки при помощи грубой силы бесполезно. Если компьютер может проверять миллион вариантов в секунду, проверка всех возможных вариантов рюкзака потребует свыше 10^{46} лет. Даже мил-

лион машин, работающих параллельно, не успеет решить эту задачу до превращения солнца в сверхновую звезду.

Безопасность метода рюкзака

Взломали криптосистему, основанную на проблеме рюкзака, не миллион машин, а пара криптографов. Сначала был раскрыт единственный бит открытого текста [725]. Затем Шамир показал, что в определенных обстоятельствах рюкзак может быть взломан [1415, 1416]. Были и другие достижения - [1428, 38, 754, 516, 488] - но никто не мог взломать систему Мартина-Хеллмана в общем случае. Наконец Шамир и Циппел (Zippel) [1418, 1419, 1421] обнаружили слабые места в преобразовании, что позволило им восстановить сверхвозрастающую последовательность рюкзака по нормальной. Точные доказательства выходят за рамки этой книги, но их хороший обзор можно найти в [1233, 1244]. На конференции, где докладывались эти результаты, вскрытие было продемонстрировано по стадиям на компьютере Apple II [492, 494].

Варианты рюкзака

После вскрытия оригинальной схемы Меркла-Хеллмана было предложено множество других систем на принципе рюкзака: несколько последовательных рюкзаков, рюкзаки Грэм-Шамира (Graham-Shamir), и другие. Все они были проанализированы и взломаны, как правило, с использованием одних и тех же криптографических методов, и их обломки были сметены со скоростного шоссе криптографии [260, 253, 269, 921, 15, 919, 920, 922, 366, 254, 263, 255]. Хороший обзор этих систем и их криптоанализ можно найти в [267, 479, 257, 268].

Были предложены и другие алгоритмы, использующие похожие идеи, но все они тоже были взломаны. Криптосистема Lu-Lee [990, 13] была взломана в [20, 614, 873], ее модификация [507] также оказалась небезопасной [1620]. Вскрытия криптосистемы Goodman-McAuley приведены в [646, 647, 267, 268]. Криптосистема Pieprzyk [1246] была взломана аналогичным образом. Криптосистема Niemi [1169], основанная на модульных рюкзаках, взломана в [345, 788]. Новый, многостадийный рюкзак [747] пока еще не был взломан, но я не оптимистичен. Другим вариантом является [294].

Хотя вариант алгоритма рюкзака в настоящее время безопасен - алгоритм рюкзака Char-Rivest [356], несмотря на "специализированное вскрытие" [743] - количество необходимых вычислений делает его намного менее полезным, чем другие рассмотренные здесь алгоритмы. Вариант, названный Powerline System (система электропитания) небезопасен [958]. Более того, учитывая легкость с которой пали все остальные варианты, доверять устоявшим пока вариантом, по видимому, неосторожно.

Патенты

Оригинальный алгоритм Меркла-Хеллмана запатентован в Соединенных Штатах [720] и в остальном мире (см. 18th). Public Key Partners (PKP) получила лицензию на патент вместе с другими патентами криптографии с открытыми ключами (см. раздел 25.5). Время действия патента США истечет 19 августа 1997 года.

Табл. 19-1.
Иностранные патенты на алгоритм рюкзака Меркла-Хеллмана

Страна	Номер	Дата получения
Бельгия	871039	5 апреля 1979 года
Нидерланды	7810063	10 апреля 1979 года
Великобритания	2006580	2 мая 1979 года
Германия	2843583	10 мая 1979 года
Швеция	7810478	14 мая 1979 года
Франция	2405532	8 июня 1979 года
Германия	2843583	3 января 1982 года
Германия	2857905	15 июля 1982 года
Канада	1128159	20 июля 1982 года
Великобритания	2.006580	18 августа 1982 года
Швейцария	63416114	14 января 1983 года
Италия	1099780	28 сентября 1985 года

19.3 RSA

Вскоре после алгоритма рюкзака Меркла появился первый полноценный алгоритм с открытым ключом , который можно использовать для шифрования и цифровых подписей : RSA [1328, 1329]. Из всех предложенных за эти годы алгоритмов с открытыми ключами RSA проще всего понять и реализовать. (Мартин Гарднер (Martin Gardner) опубликовал раннее описание алгоритма в своей колонке "Математические игры" в *Scientific American* [599].) Он также является самым популярным. Названный в честь трех изобретателей - Рона Ривеста (Ron Rivest), Ади Шамира (Adi Shamir) и Леонарда Эдлмана (Leonard Adleman) - этот алгоритм многие годы противостоит интенсивному криптоанализу. Хотя криптоанализ ни доказал, ни опроверг безопасность RSA, он, по сути, обосновывает уровень доверия к алгоритму.

Безопасность RSA основана на трудности разложения на множители больших чисел. Открытый и закрытый ключи являются функциями двух больших (100 - 200 разрядов или даже больше) простых чисел. Предполагается, что восстановление открытого текста по шифротексту и открытому ключу эквивалентно разложению на множители двух больших чисел.

Для генерации двух ключей используются два больших случайных простых числа , p и q. Для максимальной безопасности выбирайте p и q равной длины. Рассчитывается произведение:

$n = p\ q$

Затем случайным образом выбирается ключ шифрования e, такой что e и $(p\text{-}1)(q\text{-}1)$ являются взаимно простыми числами. Наконец расширенный алгоритм Эвклида используется для вычисления ключа дешифрирования d, такого что

$ed = 1 \pmod{(p\text{-}1)(q\text{-}1)}$

Другими словами

$d = e^{-1} \bmod ((p\text{-}1)(q\text{-}1))$

Заметим, что d и n также взаимно простые числа. Числа e и n - это открытый ключ, а число d - закрытый. Два простых числа p и q больше не нужны. Они должны быть отброшены, но не должны быть раскрыты.

Для шифрования сообщения m оно сначала разбивается на цифровые блоки, меньшие n (для двоичных данных выбирается самая большая степень числа 2, меньшая n). То есть, если p и q - 100-разрядные простые числа, то n будет содержать около 200 разрядов, и каждый блок сообщения m_i должен быть около 200 разрядов в длину. (Если нужно зашифровать фиксированное число блоков, их можно дополнить несколькими нулями слева, чтобы гарантировать, что блоки всегда будут меньше n. Зашифрованное сообщение c будет состоять из блоков c_i той же самой длины. Формула шифрования выглядит так

$c_i = m_i^e \bmod n$

Для расшифровки сообщения возьмите каждый зашифрованный блок c_i и вычислите

$m_i = c_i^d \bmod n$

Так как

$c_i^d = (m_i^e)^d = m_i^{ed} = m_i^{k(p\text{-}1)(q\text{-}1)+1} = m_i m_i^{k(p\text{-}1)(q\text{-}1)} = m_i * 1 = m_i;$ все $(\bmod n)$

формула восстанавливает сообщение. Это сведено в 17-й.

Табл. 19-2.
Шифрование RSA

Открытый ключ:	
n	произведение двух простых чисел p и q (p и q должны храниться в секрете)
e	число, взаимно простое с $(p\text{-}1)(q\text{-}1)$
Закрытый ключ:	
d	$e^{-1} \bmod ((p\text{-}1)(q\text{-}1))$
Шифрование:	
$c = m^e \bmod n$	
Дешифрирование:	
$m = c^d \bmod n$	

Точно также сообщение может быть зашифровано с помощью d, а зашифровано с помощью e, возможен любой выбор. Я уберегу вас от теории чисел, доказывающей, почему этот алгоритм работает. В большинстве

книг по криптографии этот вопрос подробно рассмотрен .

Короткий пример возможно поможет пояснить работу алгоритма . Если $p = 47$ и $q = 71$, то

$n = pq = 3337$

Ключ e не должен иметь общих множителей

$(p-1)(q-1)= 46*70 =3220$

Выберем (случайно) e равным 79. В этом случае $d = 79^{-1} \bmod 3220 = 1019$

При вычислении этого числа использован расширенный алгоритм Эвклида (см. раздел 11.3). Опубликуем e и n, сохранив в секрете d. Отбросим p и q. Для шифрования сообщения

m = 6882326879666683

сначала разделим его на маленькие блоки . Для нашего случая подойдут трехбуквенные блоки . Сообщение разбивается на шесть блоков m_i:

$m_1 = 688$

$m_2 = 232$

$m_3 = 687$

$m_4 = 966$

$m_5 = 668$

$m_6 = 003$

Первый блок шифруется как $688^{79} \bmod 3337 = 1570 = c_1$

Выполняя те же операции для последующих блоков, создает шифротекст сообщения :

c = 1570 2756 2091 2276 2423 158

Для дешифрирование нужно выполнить такое же возведение в степень, используя ключ дешифрирования 1019:

$1570^{1019} \bmod 3337 = 688 = m_1$

Аналогично восстанавливается оставшаяся часть сообщения .

Аппаратные реализации RSA

Существует много публикаций, затрагивающих тему аппаратных реализаций RSA [1314, 1474, 1456, 1316, 1485, 874, 1222, 87, 1410, 1409, 1343, 998, 367, 1429, 523, 772]. Хорошими обзорными статьями служат [258, 872]. Шифрование RSA выполняется многими микросхемами [1310, 252, 1101, 1317, 874, 69, 737, 594, 1275, 1563, 509, 1223]. Частичный список доступных в настоящее время микросхем RSA, взятый из [150, 258], приведен в 16th. Не все из них доступны в свободной продаже .

Табл. 19-3.
Существующие микросхемы RSA

Компания	Тактовая частота	Скорость передачи в Бодах на 512 бит	Тактовые циклы для шифрования 512 бит	Технология	Битов на микросхему	Количество транзисторов
Alpha Techn.	25 МГц	13K	0.98 M	2 микрона	1024	180000
AT&T	15 МГц	19K	0.4 M	1.5 микрона	298	100000
British Telecom	10 МГц	5.IK	1 M	2.5 микрона	256	-----
Business Sim. Ltd.	5 МГц	3.8K	0.67 M	Вентильная матрица	32	-----
CalmosSyst-Inc.	20 МГц	2.8K	0.36 M	2 микрона	593	95000
CNET	25 МГц	5.3K	2.3 M	1 микрон	1024	100000
Cryptech	14 МГц	17K	0.4 M	Вентильная матрица	120	33000
Cylink	30 МГц	6.8K	1.2 M	1.5 микрона	1024	150000
GEC Marconi	25 МГц	10.2K	0.67 M	1.4 микрона	512	160000
Pijnenburg	25 МГц	50K	0.256 M	1 микрон	1024	400000
Sandia	8 МГц	IOK	0.4 M	2 микрона	272	86000
Siemens	5 МГц	8.5K	0.03 M	1 микрон	512	60000

Скорость RSA

Аппаратно RSA примерно в 1000 раз медленнее DES. Скорость работы самой быстрой СБИС-реализации RSA с 512-битовым модулем - 64 килобита в секунду [258]. Существуют также микросхемы, которые выполня-

ют 1024-битовое шифрование RSA. В настоящее время разрабатываются микросхемы, которые, используя 512-битовый модуль, приблизятся к рубежу 1 Мбит/с. Возможно, они появятся в 1995 году. Производители также применяют RSA в интеллектуальных карточках, но эти реализации медленнее.

Программно DES примерно в 100 раз быстрее RSA. Эти числа могут незначительно измениться при изменении технологии, но RSA никогда не достигнет скорости симметричных алгоритмов. В 15-й приведены примеры скоростей программного шифрования RSA [918].

Табл. 19-4.
Скорости RSA для различных длин модулей при 8-битовом открытом ключе (на SPARC II)

	512 битов	768 битов	1024 бита
Шифрование	0.03 с	0.05 с	0.08 с
Дешифрирование	0.16 с	0.48 с	0.93 с
Подпись	0.16 с	0.52 с	0.97 с
Проверка	0.02 с	0.07 с	0.08 с

Программные Speedups

Шифрование RSA выполняется намного быстрей, если вы правильно выберете значение e. Тремя наиболее частыми вариантами являются 3, 17 и 65537 ($2^{16} + 1$). (Двоичное представление 65537 содержит только две единицы, поэтому для возведения в степень нужно выполнить только 17 умножений.) X.509 советует 65537 [304], PEM рекомендует 3 [76], а PKCS #l (см. раздел 24.14) - 3 или 65537 [1345]. Не существует никаких проблем безопасности, связанных с использованием в качестве e любого из этих трех значений (при условии, что вы дополняете сообщения случайными числами - см. раздел ниже), даже если одно и то же значение e используется целой группой пользователей.

Операции с закрытым ключом можно ускорить при помощи китайской теоремы от остатках, если вы сохранили значения p и q, а также дополнительные значения: $d \bmod (p - 1)$, $d \bmod (q - 1)$ и $q^{-1} \bmod p$ [1283, 1276]. Эти дополнительные числа можно легко вычислить по закрытому и открытому ключам.

Безопасность RSA

Безопасность RSA полностью зависит от проблемы разложения на множители больших чисел. Технически, это утверждение о безопасности лживо. Предполагается, что безопасность RSA зависит от проблемы разложения на множители больших чисел. Никогда не было доказано математически, что нужно разложить n на множители, чтобы восстановить m по c и e. Понятно, что может быть открыт совсем иной способ криптоанализа RSA. Однако, если этот новый способ позволит криптоаналитику получить d, он также может быть использован для разложения на множители больших чисел. Я не слишком волнуюсь об этом.

Также можно вскрыть RSA, угадав значение $(p-1)(q-1)$. Это вскрытие не проще разложения n на множители [1616].

Для сверхскептиков: доказано, что некоторые варианты RSA также сложны, как и разложение на множители (см. раздел 19.5). Загляните также в [361, где показано, что раскрытие даже нескольких битов информации по зашифрованному RSA шифротексту не легче, чем дешифрирование всего сообщения.

Самым очевидным средством вскрытия является разложение n на множители. Любой противник сможет получить открытый ключ e и модуль n. Чтобы найти ключ дешифрирования d, противник должен разложить n на множители. Современное состояние технологии разложения на множители рассматривалось в разделе 11.4. В настоящее время передним краем этой технологии является число, содержащее 129 десятичных цифр. Значит, n должно быть больше этого значения. Рекомендации по выбору длины открытого ключа приведены в разделе 7.2.

Конечно, криптоаналитик может перебирать все возможные d, пока он не подберет правильное значение. Но такое вскрытие грубой силой даже менее эффективно, чем попытка разложить n на множители.

Время от времени появляются заявления о том, что найден простой способ вскрытия RSA, но пока ни одно из подобных заявлений не подтвердилось. Например, в 1993 году в черновике статьи Вильяма Пейна (William Payne) был предложен метод, основанный на малой теореме Ферма [1234]. К сожалению, этот метод оказался медленнее разложения на множители

Существует еще один повод для беспокойства. Большинство общепринятых алгоритмов вычисления простых чисел p и q вероятностны, что произойдет, если p или q окажется составным? Ну, во первых, можно свести вероятность такого события до нужного минимума. И даже если это произойдет, скорее всего такое событие будет

сразу же обнаружено - шифрование и дешифрирование не будут работать. Существует ряд чисел, называемых числами Кармайкла (Carmichael), которые не могут обнаружить определенные вероятностные алгоритмы поиска простых чисел. Они небезопасны, но чрезвычайно редки [746]. Честно говоря, меня бы это не обеспокоило.

Вскрытие с выбранным шифротекстом против RSA

Некоторые вскрытия работают против реализаций RSA. Они вскрывают не сам базовый алгоритм, а надстроенный над ним протокол. Важно понимать, что само по себе использование RSA не обеспечивает безопасности. Дело в реализации.

Сценарий 1: Еве, подслушавшей линии связи Алисы, удалось перехватить сообщение c, шифрованное с помощью RSA открытым ключом Алисы. Ева хочет прочитать сообщение. На языке математики, ей нужно m, для которого

$m = c^d$

Для раскрытия m она сначала выбирает первое случайное число r, меньшее n. Она достает открытый ключ Алисы e. Затем она вычисляет

$x = r^e \bmod n$

$y = xc \bmod n$

$t = r^{-1} \bmod n$

Если $x = r^e \bmod n$, то $r = x^d \bmod n$.

Теперь просит Алису подписать y ее закрытым ключом, таким образом расшифровав y. (Алиса должна подписать сообщение, а не его хэш сумму.) Не забывайте, Алиса никогда раньше не видела y. Алиса посылает Еве

$u = y^d \bmod n$

Теперь Ева вычисляет

$tu \bmod n = r^{-1} y^d \bmod = r^{-1} x^d c^d \bmod n = c^d \bmod n = m$

И Ева получает m.

Сценарий 2: Трент - это компьютер-нотариус. Если Алиса хочет заверить документ, она посылает его Тренту. Трент подписывает его цифровой подписью RSA и отправляет обратно. (Однонаправленные хэш-функции не используются, Трент шифрует все сообщение своим закрытым ключом.)

Мэллори хочет, чтобы Трент подписал такое сообщение, которое в обычном случае он он никогда не подпишет. Может быть это фальшивая временная метка, может быть автором этого сообщения является другое лицо. Какой бы ни была причина, Трент никогда не подпишет это сообщение, если у него будет возможность выбора. Назовем это сообщение m'.

Сначала Мэллори выбирает произвольное значение x и вычисляет $y = x^e \bmod n$. e он может получить без труда - это открытый ключ Трента, который должен быть опубликован, чтобы можно было проверять подписи Трента. Теперь Мэллори вычисляет $m = ym' \bmod n$ и посылает m Тренту на подпись. Трент возвращает $m^d \bmod n$. Now Мэллори вычисляет $(m^d \bmod n)x^{-1} \bmod n$, которое равно $n^d \bmod n$ и является подписью m'.

На самом деле Мэллори может использовать множество способов решить подобную задачу [423, 458, 486]. Слабым местом, которое используют такие вскрытия, является сохранение мультипликативной структуры входа при возведении в степень. То есть:

$(xm)^d \bmod n = x^d m^d \bmod n$

Сценарий 3: Ева хочет, чтобы Алиса подписала m_3. Она создает два сообщения, m_1 и m_2, такие что

$m_3 = m_1 m_2 \pmod n$

Если Ева сможет заставить Алису подписать m_1 и m_2, она может вычислить подпись для m_3:

$m_3^d = (m_1^d \bmod n)\ (m_2^d \bmod n)$

Мораль: Никогда не пользуйтесь алгоритмом RSA для подписи случайных документов, подсунутых вам посторонними. Всегда сначала воспользуйтесь однонаправленной хэш-фунцией. Формат блоков ISO 9796 предотвращает это вскрытие.

Вскрытие общего модуля RSA

При реализации RSA можно попробовать раздать всем пользователям одинаковый модуль n, но каждому свои значения показателей степени e и d. К сожалению, это не работает. Наиболее очевидная проблема в том,

что если одно и то же сообщение когда-нибудь шифровалось разными показателями степени (с одним и тем же модулем), и эти два показателя - взаимно простые числа (как обычно и бывает), то открытый текст может быть раскрыт, даже не зная ни одного ключа дешифрирования [1457].

Пусть m - открытый текст сообщения. Два ключа шифрования - e_1 и e_2. Общий модуль - n. Шифротекстами сообщения являются:

$$c_1 = m^{e_1} \bmod n$$

$$c_2 = m^{e_2} \bmod n$$

Криптоаналитик знает n, e_1, e_2, c_1 и c_2. Вот как он узнает m.

Так как e_1 и e_2 - взаимно простые числа, то с помощью расширенного алгоритма Эвклида r и s, для которых

$$re_1 + se_2 = 1$$

Считая r отрицательным (или r, или s должно быть отрицательным, пусть отрицательным будет r), то снова можно воспользоваться расширенным алгоритмом для вычисления c_1^{-1}. Затем

$$(c_1^{-1})^{-r} * c_2^{s} = m \bmod n$$

Существует два других, более тонких вскрытия систем такого типа. Одно использует вероятностный метод для разложения n на множители. Другой - детерминированный алгоритм вычисления какого-нибудь секретного ключа без разложения модуля на множители. Оба вскрытия подробно описаны в [449].

Мораль: Не делайте n общим для группы пользователей.

Вскрытие малого показателя шифрования RSA

Шифрование и проверка подписи RSA выполняется быстрее, если для e используется небольшое значение, но это также может быть небезопасным [704]. Если $e(e + 1)/2$ линейно зависящих сообщений с различными открытыми ключами шифруются одним и тем же значением e, существует способ вскрыть такую систему. Если сообщений не так много, или если сообщения не связаны, то проблем нет. Если сообщения одинаковы, то достаточно e сообщений. Проще всего дополнять сообщения независимыми случайными числами.

Это также гарантирует, что $m^e \bmod n \neq m^e$. Так делается в большинстве практических реализаций RSA, например, в PEM и PGP (см. разделы 24.10 и 24.12).

Мораль: Дополняйте сообщения перед шифрованием случайными значениями, убедитесь, что размер m примерно равен n.

Вскрытие малого показателя дешифрирования RSA

Другим вскрытием, предложенным Майкл Винер (Michael Wiener), раскрывает d, где d не превышает четверти размера n, а e меньше n [1596]. При случайном выборе e и d это встречается редко, и никогда не произойдет, если значение e мало.

Мораль: Выбирайте большое значение d.

Полученные уроки

Джудит Мур (Judith Moore) на основании перечисленных вскрытий приводит следующие ограничения RSA [1114, 1115]:

— Знание одной пары показателей шифрования/дешифрирования для данного модуля позволяет взломщику разложить модуль на множители.

— Знание одной пары показателей шифрования/дешифрирования для данного модуля позволяет взломщику вычислить другие пары показателей, не раскладывая модуль на множители.

— В протоколах сетей связи, применяющих RSA, не должен использоваться общий модуль. (Это является быть очевидным следствием предыдущих двух пунктов.)

— Для предотвращения вскрытия малого показателя шифрования сообщения должны быть дополнены случайными значениями.

— Показатель дешифрирования должен быть большим.

Не забывайте, недостаточно использовать безопасный криптографический алгоритм, должны быть безопасными вся криптосистема и криптографический протокол. Слабое место любого из трех этих компонентов сделает небезопасной всю систему.

Вскрытие шифрования и подписи с использованием RSA

Имеет смысл подписывать сообщение перед шифрованием (см. раздел 2.7), но на практике никто не выполняет этого. Для RSA можно вскрыть протоколы, шифрующие сообщение до его подписания [48].

Алиса хочет послать сообщение Бобу. Сначала она шифрует его открытым ключом Боба, а затем подписывает своим закрытым ключом. Ее зашифрованное и подписанное сообщение выглядит так:

$$m^{e_B} \bmod n_B)^{d_A} \bmod n_A$$

Вот как Боб может доказать, что Алиса послала ему m', а не m. Так как Бобу известно разложение на множители n_B (это его собственный модуль), он может вычислить дискретные логарифмы по основанию n_B. Следовательно, ему нужно только найти x, для которого

$$m^x = m \bmod n_B$$

Тогда, если он может опубликовать xe_B в качестве своего нового открытого показателя степени и сохранить свой прежний модуль n_B, он сможет утверждать, что Алиса послала ему сообщение m', зашифрованное этим новым показателем.

В некоторых случаях это особенно неприятное вскрытие. Заметим, что хэш-функции не решают проблему. Однако она решается при использовании для каждого пользователя фиксированного показ ателя шифрования.

Стандарты

RSA *de facto* является стандартом почти по всему миру. ISO почти, but not quite, created an RSA digital-signature standard; RSA служит информационным дополнением ISO 9796 [762.]. Французское банковское сообщество приняло RSA в качестве стандарта [525], так же поступили и австралийцы [1498]. В Соединенных Штатах из-за давления NSA и патентных вопросов в настоящее время нет стандарта для шифрования с открытым ключом. Многие американские компании используют PKCS (см. раздел 24.14), написанный RSA Data Security, Inc. RSA определен и в качестве чернового банковского стандарта ANSI [61].

Патенты

Алгоритм RSA запатентован в Соединенных Штатах [1330], но ни водной другой стране. PKP получила лицензию вместе с другими патентами в области криптографии с открытыми ключами (раздел 25.5). Срок действия патента США истекает 20 сентября 2000 года.

19.4 Pohlig-Hellman

Схема шифрования Pohlig-Hellman [1253] похожа на RSA. Это не симметричный алгоритм, так как для шифрования и дешифрирования используются различные ключи. Это не схема с открытым ключом, потому что ключи легко получаются один из другого, и ключ шифрования, и ключ дешифрирования должны храниться в секрете. Как и в RSA,

$$C = P^e \bmod n$$

$$P = C^d \bmod n$$

где

$ed \equiv 1$ (mod какое-нибудь составное число)

В отличие от RSA n не определяется с помощью двух простых чисел и остается частью закрытого ключа. Если у кого-нибудь есть e и n, он может вычислить d. Не зная e или d, противник будет вынужден вычислить

$$e = \log_p C \bmod n$$

Мы уже видели, что это является трудной проблемой.

Патенты

Алгоритм Pohlig-Hellman запатентован в США [722] и в Канаде. PKP получила лицензию вместе с другими патентами в области криптографии с открытыми ключами (см. раздел 25.5).

19.5 Rabin

Безопасность схемы Рабина (Rabin) [1283, 1601] опирается на сложность поиска квадратных корней по модулю составного числа. Эта проблема аналогична разложению на множители. Вот одна из реализаций этой схемы.

Сначала выбираются два простых числа p и q, конгруэнтных 3 mod 4. Эти простые числа являются закрытым ключом, а их произведение $n = pq$ - открытым ключом.

Для шифрования сообщения M (M должно быть меньше n), просто вычисляется

$C = M^2 \bmod n$

Дешифрирование сообщения также несложно, но немного скучнее. Так как получатель знает p и q, он может решить две конгруэнтности с помощью китайской теоремы об остатках. Вычисляется

$m_1 = C^{(p+1)/4} \bmod p$

$m_2 = (p - C^{(p+1)/4}) \bmod p$

$m_3 = C^{(q+1)/4} \bmod q$

$m_4 = (q - C^{(q+1)/4}) \bmod q$

Затем выбирается целые числа $a = q(q^{-1} \bmod p)$ и $b = p(p^{-1} \bmod q)$. Четырьмя возможными решениями являются:

$M_1 = (am_1 + bm_3) \bmod n$

$M_2 = (am_1 + bm_4) \bmod n$

$M_3 = (am_2 + bm_3) \bmod n$

$M_4 = (am_2 + bm_4) \bmod n$

Один из четырех результатов, M_1, M_2, M_3 и M_4, равно M. Если сообщение написано по английски, выбрать правильное M_i нетрудно. С другой стороны, если сообщение является потоком случайных битов (скажем, для генерации ключей или цифровой подписи), способа определить, какое M_i - правильное, нет. Одним из способов решить эту проблему служит добавление к сообщению перед шифрованием известного заголовка.

Williams

Хью Вильямс (Hugh Williams) переопределил схему Рабина, чтобы устранить эти недостатки [1601]. В его схеме p и q выбираются так, чтобы

$p \equiv 3 \bmod 8$

$q \equiv 7 \bmod 8$

и

$N = pq$

Кроме того, используется небольшое целое число, S, для которого $J(S,N) = -1$. (J - это символ Якоби - см. раздел I I .3). N и S опубликовываются. Секретным ключом является k, для которого

$k = 1/2\ (1/4\ (p - 1)\ (q - 1) + 1)$

Для шифрования сообщения M вычисляется c_1, такое что $J(M,N) = (-1)^{c_1}$. Затем вычисляется $M' = (S^{c_1} * M) \bmod N$. Как и в схеме Рабина, $C = M'^2 \bmod N$. И $c_2 = M' \bmod 2$. Окончательным шифротекстом сообщения является тройка:

(C, c_1, c_2)

Для дешифрирования C, получатель вычисляет M'' с помощью

$C^k \equiv \pm M'' \pmod N$

Правильный знак M'' определяет c_2. Наконец

$M = (S^{c_1} * (-1)^{c_1} * M'') \bmod N$

Впоследствии Вильямс улучшил эту схему в [1603, 1604, 1605]. Вместо возведения в квадрат открытого текста сообщения, возведите его в третью степени. Большие простые числа должны быть конгруэнтны 1 по модулю 3, иначе открытый и закрытый ключи окажутся одинаковыми. Даже лучше, существует только одна уникальная расшифровка каждого шифрования.

Преимущество схем Рабина и Вильямса перед RSA в том, что доказано, что они также безопасны, как и разложение на множители. Однако перед вскрытием с выбранным шифротекстом они совершенно беззащитны. Если вы собираетесь использовать эти схемы для случаев, когда взломщик может выполнить такое вскрытие (например, алгоритм цифровой подписи, когда взломщик может выбирать подписываемые сообщения), не за-

бывайте использовать перед подписанием однонаправленную хэш-функцию. Рабин предложил другой способ защититься от такого вскрытия: к каждому сообщению перед хэшированием и подписанием добавляется уникальная случайная строка. К несчастью, после добавления однонаправленной хэш-функцией тот факт, что система столь же безопасна, как и разложение на множители, больше не является доказанным [628]. Хотя с практической точки зрения добавление хэширования не может ослабить систему.

Другими вариантами схемы Рабина являются [972, 909, 696, 697, 1439, 989]. Двумерный вариант описан в [866, 889].

19.6 ElGamal

Схему ElGamal [518,519] можно использовать как для цифровых подписей, так и для шифрования, его безопасность основана на трудности вычисления дискретных логарифмов в конечном поле.

Для генерации пары ключей сначала выбирается простое число p и два случайных числа, g и x, оба эти числа должны быть меньше p. Затем вычисляется

$y = g^x \bmod p$

Открытым ключом являются y, g и p. И g, и p можно сделать общими для группы пользователей. Закрытым ключом является x.

Подписи ElGamal

Чтобы подписать сообщение M, сначала выбирается случайное число k, взаимно простое с p-1. Затем вычисляется

$a = g^k \bmod p$

и с помощью расширенного алгоритма Эвклида находится b в следующем уравнении:

$M = (xa + kb) \bmod (p - 1)$

Подписью является пара чисел: a и b. Случайное значение k должно храниться в секрете. Для проверки подписи нужно убедиться, что

$y^a a^b \bmod p = g^M \bmod p$

Каждая подпись или шифрование ElGamal требует нового значения k, и это значение должно быть выбрано случайным образом. Если когда-нибудь Ева раскроет k, используемое Алисой, она сможет раскрыть закрытый ключ Алисы x. Если Ева когда-нибудь сможет получить два сообщения, подписанные или зашифрованные с помощью одного и того же k, то она сможет раскрыть x, даже не зная значение k. Описание ElGamal сведено в 14-й.

Табл. 19-5.
Подписи ElGamal

Открытый ключ:	
p	простое число (может быть общим для группы пользователей)
g	$<p$ (может быть общим для группы пользователей)
y	$=g^x \bmod p$
Закрытый ключ:	
x	$<p$
Подпись:	
k	выбирается случайным образом, взаимно простое с p-1
a	(подпись) $=g^k \bmod p$
b	(подпись), такое что $M = (xa + kb) \bmod (p - 1)$
Проверка:	
Подпись считается правильной, если $y^a a^b \bmod p = g^M \bmod p$	

Например, выберем $p = 11$ и $g = 2$, а закрытый ключ $x = 8$. Вычислим

$y = g^x \bmod p = 2^8 \bmod 11 = 3$

Открытым ключом являются $y = 3$, $g = 2$ и $p = 11$. Чтобы подписать $M = 5$, сначала выберем случайное число k=9. Убеждаемся, что gcd(9, 10)= 1. Вычисляем

$a = g^k \bmod p = 2^9 \bmod 11 = 6$

и с помощью расширенного алгоритма Эвклида находим b:

$M = (xa + kb) \bmod (p - 1)$

$5 = (8*6 + 9*b) \bmod 10$

Решение: $b = 3$, а подпись представляет собой пару: $a = 6$ и $b = 3$.

Для проверки подписи убедимся, что

$y^a a^b \bmod p = g^M \bmod p$

$3^6 6^3 \bmod 11 = 2^5 \bmod 11$

Вариант ElGamal, используемый для подписей, описан в [1377]. Томас Бет (Thomas Beth) изобрел вариант схемы ElGamal, подходящий для доказательства идентичности [146]. Существуют варианты для проверки подлинности пароля [312] и для обмена ключами [773]. И еще тысячи и тысячи других (см. раздел 20.4).

Шифрование ElGamal

Модификация ElGamal позволяет шифровать сообщения. Для шифрования сообщения M сначала выбирается случайное число k, взаимно простое с p - 1. Затем вычисляются

$a = g^k \bmod p$

$b = y^k M \bmod p$

Пара (a,b) является шифротекстом. Обратите внимание, что шифротекст в два раза длиннее открытого текста. Для дешифрирования (a,b) вычисляется

$M = b/a^x \bmod p$

Так как $a^x \equiv g^{kx} \pmod p$ и $b/a^x \equiv y^k M/a^x \equiv g^{xk} M/ g^{kx} = M \pmod p$, то все работает (см. 13-й). По сути это то же самое, что и обмен ключами Диффи-Хеллмана (см. раздел 22.1) за исключением того, что y - это часть ключа, а при шифровании сообщение умножается на y^k.

Табл. 19-6.
Шифрование ElGamal

	Открытый ключ:
p	простое число (может быть общим для группы пользователей)
g	$<p$ (может быть общим для группы пользователей)
y	$=g^x \bmod p$
	Закрытый ключ:
x	$<p$
	Шифрование:
k	выбирается случайным образом, взаимно простое с p-1
a	(шифротекст) $=g^k \bmod p$
b	(шифротекст)$= y^k M \bmod p$
	Дешифрирование:
M (открытый текст)	$= b/a^x \bmod p$

Скорость

Некоторые примеры скорости работы программных реализаций ElGamal приведены в 12-й [918].

Табл. 19-7.

Скорости ElGamal для различных длин модулей при 160-битовом показателе степени (на SPARC II)

	512 битов	768 битов	1024 битов
Шифрование	0.33 с	0.80 с	1.09 с
Дешифрирование	0.24 с	0.58 с	0.77 с
Подпись	0.25 с	0.47 с	0.63 с
Проверка	l.37 с	5.12 с	9.30 с

Патенты

ElGamal незапатентован. Но, прежде чем двигаться вперед и реализовывать алгоритм, нужно знать, что PKP считает, что этот алгоритм попадает под действие патента Диффи-Хеллмана [718]. Однако срок действия патента Диффи-Хеллмана заканчивается 29 апреля 1997 года, что делает ElGamal первым криптографическим плгоритмом с открытыми ключами, пригодным для шифрования и цифровых подписей и несвязанным в Соедине нных Штатах патентами. Я не могу дождаться этого момента.

19.7 McEliece

В 1978 году Роберт МакЭлис (Robert McEliece) разработал криптосистему с открытыми ключами на основе теории алгебраического кодирования [1041]. Этот алгоритм использует существование определенного класса исправляющих ошибки кодов, называемых **кодами Гоппа** (Goppa). Он предлагал создать код Гоппа и замаскировать его как обычный линейный код. Существует быстрый алгоритм декодирования кодов Гоппа, но общая проблема найти слово кода по данному весу в линейном двоичном коде является **NP-полной**. Хорошее описание этого алгоритма можно найти в [1233], см. также [1562]. Ниже приведен только краткий обзор.

Пусть $d_H(x,y)$ обозначает расстояние Хэмминга между x и y. Числа n, k и t служат параметрами системы.

Закрытый ключ состоит из трех частей: G' - это матрица генерации года Гоппа, исправляющего t ошибок. P - это матрица перестановок размером $n*n$. S - это nonsingular матрица размером $k*k$.

Открытым ключом служит матрица G размером $k*n$: $G = SG'P$.

Открытый текст сообщений представляет собой строку k битов в виде k-элементного вектора над полем GF(2).

Для шифрования сообщения случайным образом выбирается n-элементный вектор z над полем GF(2), для которого расстояние Хэмминга меньше или равно t.

$c = mG + z$

Для дешифрирования сообщения сначала вычисляется $c' = cP^{-1}$. Затем с помощью декодирующего алгоритма для кодов Гоппа находится m', для которого $d_H(m'G,c)$ меньше или равно t. Наконец вычисляется $m = m'S^{-1}$.

В своей оригинальной работе МакЭлис предложил значения n = 1024, t = 50 и k = 524. Это минимальные значения, требуемые для безопасности.

Хотя этот алгоритм был одним из первых алгоритмов с открытыми ключами, и вне появлялось публикаций о его успешном криптоаналитическом вскрытии, он не получил широкого признания в криптографическом с ообществе. Схема на два-три порядка быстрее, чем RSA, но у нее есть ряд недостатков. Открытый ключ огромен: 2^{19} битов. Сильно увеличивается объем данных - шифротекст в два раза длиннее открытого текста.

Ряд попыток криптоанализа этой системы можно найти в [8, 943, 1559, 306]. Ни одна из них не достигла успеха для общего случая, хотя сходство между алгоритмом МакЭлиса и алгоритмом рюкзака немного волнует.

В 1991 два русских криптографа заявили, что взломали систему МакЭлиса с некоторыми параметрами [882]. В их статье это утверждение не было обосновано, и большинство криптографов не приняли во внимание этот результат. Еще одно выполненное русскими вскрытие, которое нельзя непосредственно использовать против системы МакЭлиса, описано в [1447, 1448]. Расширения McEliece можно найти в [424, 1227, 976].

Другие алгоритмы, основанные на линейных кодах, исправляющих ошибки

Алгоритм Нидеррейтера (Niederreiter) [1167] очень близок к алгоритму МакЭлиса и считает, что открытый ключ - это случайная матрица проверки четности кода, исправляющего ошибки. Закрытым ключом служит эффективный алгоритм декодирования этой матрицы.

Другой алгоритм, используемый для идентификации и цифровых подписей, основан на декодировании

синдрома [1501], пояснения см. в [306]. Алгоритм [1621], использующий коды, исправляющие ошибки, небезопасен [698, 33, 31, 1560, 32].

19.8 Криптосистемы с эллиптическими кривыми

Эллиптические кривые изучались многие годы, и по этому вопросу существует огромное количество литературы. В 1985 году Нил Коблиц (Neal Koblitz) и В.С. Миллер (V. S. Miller) независимо предложили использовать их для криптосистем с открытыми ключами [867, 1095]. Они не изобрели нового криптографического алгоритма, использующего эллиптические кривые над конечными полями, но реализовали существующие алгоритмы, подобные Diffie-Hellman, с помощью эллиптических кривых.

Эллиптические кривые вызывают интерес, потому что они обеспечивают способ конструирования "элементов" и "правил объединения", образующих группы. Свойства этих групп известны достаточно хорошо, чтобы использовать их для криптографических алгоритмов, но у них нет определенных свойств, облегчающих криптоанализ. Например, понятие "гладкости" неприменимо к эллиптическим кривым. То есть, не существует такого множества небольших элементов, используя которые с помощью простого алгоритма с высокой вероятностью можно выразить случайный элемент. Следовательно, алгоритмы вычисления дискретного логарифма показателя степени не работают work. Подробности см. в [1095].

Особенно интересны эллиптические кривые над полем $GF(2^n)$. Для n в диапазоне от 130 до 200 несложно разработать схему и относительно просто реализовать арифметический процессор для используемого поля. Такие алгоритмы потенциально могут послужить основой для более быстрых криптосистем с открытыми ключами и меньшими размерами ключей. С помощью эллиптических кривых над конечными полями могут быть реализованы многие алгоритмы с открытыми ключами, такие как Diffie-Hellman, EIGamal и Schnorr.

Соответствующая математика сложна и выходит за рамки этой книги. Интересующимся этой темой я предлагаю прочитать две вышеупомянутые работы и отличную книгу Альфреда Менезеса (Alfred Menezes) [1059]. Эллиптические кривые используются двумя аналогами RSA [890, 454]. Другими работами являются [23, 119, 1062, 869, 152, 871, 892, 25, 895, 353, 1061, 26, 913, 914, 915]. Криптосистемы с ключами небольшой длины на базе эллиптических кривых рассматриваются в [701]. Алгоритм Fast Elliptic Encryption (FEE, быстрое эллиптическое шифрование) компании Next Computer Inc. также использует эллиптические кривые [388]. Приятной особенностью FEE является то, что закрытый ключ может быть любой легко запоминающейся строкой. Предлагаются и криптосистемы, использующие гиперэллиптические кривые [868, 870, 1441, 1214].

19.9 LUC

Некоторые криптографы разработали обобщенные модификации RSA, которые используют различные перестановочные многочлены вместо возведения в степень. Вариант, называющийся Kravitz-Reed и использующий неприводимые двоичные многочлены [898], небезопасен [451, 589]. Винфрид Мюллер (Winfried Müller) и Вилфрид Нобауер (Wilfried Nöbauer) используют полиномы Диксона (Dickson) [1127, 1128, 965]. Рудольф Лидл (Rudolph Lidl) и Мюллер обобщили этот подход в [966, 1126] (этот вариант назван схемой Réidi), и Нобауер проанализировал его безопасность в [1172, 1173]. (Соображения по поводу генерации простых чисел с помощью функций Лукаса (Lucas) можно найти в [969, 967, 968, 598].) Несмотря на все предыдущие разработки группе исследователей из Новой Зеландии удалось запатентовать эту схему в 1993 году, назвав ее LUC [1486, 521, 1487].

n-ое число Лукаса, $V_n(P,1)$, определяется как

$V_n(P,1) = PV_{n-1}(P,1) - V_{n-2}(P,1)$

Теория чисел Лукаса достаточно велика, и я ее пропущу. Теория последовательностей Лукаса хорошо изложена в [1307, 1308]. Особенно хорошо математика LUC описана в [1494, 708].

В любом случае для генерации пары открытый ключ/закрытый ключ сначала выбираются два больших числа p и q. Вычисляется n, произведение p и q. Ключ шифрования e - это случайное число, взаимно простое с p-1, q-1, p+1 и q+1. Существует четыре возможных ключа дешифрирования,

$d = e^{-1} \bmod (\text{НОК}(p+1), (q+1)))$

$d = e^{-1} \bmod (\text{НОК}(p+1), (q-1)))$

$d = e^{-1} \bmod (\text{НОК}(p-1), (q+1)))$

$d = e^{-1} \bmod (\text{НОК}(p-1), (q-1)))$

где НОК означает наименьшее общее кратное.

Открытым ключом являются d и n; закрытым ключом - e и n. p и q отбрасываются.

Для шифрования сообщения P (P должно быть меньше n) вычисляется

$C = V_e(P,1) \pmod{n}$

А для дешифрирования:

$P = V_d(P, 1) \pmod{n}$, с соответствующим d

В лучшем случае LUC не безопаснее RSA. А недавние, только что опубликованные результаты показывают, как взломать LUC по крайней мере в нескольких реализациях. Я не доверяю этому алгоритму.

19.10 Криптосистемы с открытым ключом на базе конечных автоматов

Китайский криптограф Тао Ренжи разработал алгоритм с открытым ключом, основанный на использовании конечных автоматов [1301, 1302, 1303, 1300, 1304, 666]. Такой же сложной задачей, как и разложение на множители произведения двух больших простых чисел, является задача разложения на составляющие произведения двух конечных автоматов. Это тем более верно, если один из автоматов нелинеен.

Большая часть работы в этой области была выполнена в Китае в 80-х годах и опубликована на китайском языке. Ренжи начал писать по английски. Его главным результатом было то, что обратное значение некоторых нелинейных (квазилинейных) автоматов является слабым тогда и только тогда, когда эти автоматы обладают определенной ступенчатой матричной структурой. Это свойство исчезает, если они объединены с другим автоматом (хотя бы линейным). В алгоритме с открытым ключом секретный ключ является инвертируемым квазилинейным автоматом, а соответствующий открытый ключ может быть получен с помощью их почленного перемножения. Данные шифруются, проходя через линейный автомат, а дешифрируются, проходя через обратные значения компонентов алгоритма (в некоторых случаях автоматы должны быть установлены в подходящее начальное значение). Эта схема работает и для шифрования, и для цифровых подписей.

О производительности таких систем вкратце можно сказать следующее: они, как и система McEliece, намного быстрее RSA, но требуют использования более длинных ключей. Длина ключа, обеспечивающая, как думают, безопасность, аналогичную 512-битовому RSA, равна 2792 битам, а 1024-битовому RSA - 4152 битам. В первом случае система шифрует данные со скоростью 20869 байт/с и дешифрирует данные со скоростью17117 байт/с, работая на 80486/33 МГц.

Ренжи опубликовал три алгоритма. Первым был FAPKC0. Эта слабая система использует линейные компоненты и, главным образом, является иллюстративной. Каждая из двух серьезных систем, FAPKC1 и FAPKC2, использует один линейный и один нелинейный компонент. Последняя сложнее, она была разработана для поддержки операции проверки подлинности.

Что касается их надежности, в Китае немало занимались этой проблемой (где сейчас свыше 30 институтов, публикующих работы по криптографии и безопасности). Из достаточного количества источников на китайском языке можно видеть, что проблема была изучена.

Привлекательной особенностью FAPKC1 и FAPKC2 является то, что они не ограждены никакими патентами США. Следовательно, так как срок действия патента на алгоритм Diffie-Hellman истекает в 1997 году, эти алгоритмы несомненно являются очень интересными.

Глава 20
Алгоритмы цифровой подписи с открытым ключом

20.1 Алгоритм цифровой подписи (DIGITAL SIGNATURE ALGORITHM, DSA)

В августе 19991 года Национальный институт стандартов и техники (National Institute of Standards and Technology, NIST) предложил для использования в своем Стандарте цифровой подписи (Digital Signature Standard, DSS) Алгоритм цифровой подписи (Digital Signature Algorithm, DSA). Согласно *Federal Register* [538]:

> Предлагается Федеральный стандарт обработки информации (Federal Information Processing Standard, FIPS) для Стандарта цифровой подписи (Digital Signature Standard, DSS). В этом стандарте определяется алгоритм цифровой подписи с открытым ключом (DSA), пригодный для федеральных применений, требующих цифровой подписи. Предложенный DSS использует открытый ключ для проверки получателем целостности полученных данных и личности отправителя. DSS также может быть использован третьей стороной для проверки правильности подписи и связанных с ней данных.
>
> В этом стандарте принимается схема подписи с открытым ключом, использующая пару преобразований для создания и проверки цифрового значения, называемого подписью.

И:

> Предложенный стандарт представляет собой результат оценки различных методик цифровой подписи. Принимая решение, NIST следовал положению раздела 2 Акта о компьютерной безопасности (Computer Security Act) 1987 года о том, что NIST разрабатывает стандарты," . . . обеспечивающие рентабельные безопасность и секретность Федеральной информации, выбирая из технологий, предлагающих сравнимую степень защиты, ту, которая обладает наиболее подходящими рабочими и эксплуатационными характеристиками".
>
> Среди факторов, рассмотренных в процессе принятия решения были уровень обеспечиваемой безопасности, простота аппаратной и программной реализации, простота экспорта за пределы США, применимость патентов, влияние на национальную безопасность и обеспечение правопорядка, а также степень эффективностикак функции подписи, так и функции проверки. Казалось, что обеспечить соответствующую защиту Федеральным системам можно многими способами. Выбранный удовлетворяет следующим требованиям:
>
> NIST ожидает, что его можно будет использовать бесплатно. Широкое использование этой технологии, обусловленной его доступностью, послужит к экономической выгоде правительства и общества.
>
> Выбранная технология обеспечивает эффективное использование операций подписи в приложениях, связанных с использованием интеллектуальных карточек. В этих приложениях операции подписи выполняются в слабой вычислительной среде интеллектуальных карточек, а процесс проверки реализуется в более мощной вычислительной среде, например, на персональном компьютере, в аппаратном криптографическом модуле или на компьютере-мэйнфрейме.

Прежде, чем все совсем запутается, позвольте мне разобраться с названиями: DSA - это алгоритм, а DSS стандарт. Стандарт использует алгоритм. Алгоритм является частью стандарта.

Реакция на заявление

Заявление NIST вызвало поток критических замечаний и обвинений. К сожалению, они были скорее политическими, чем научными. RSA Data Security, Inc., продающая алгоритм RSA, возглавила критиков DSS. Они требовали, чтобы в стандарт использовался алгоритм RSA. RSADSI получило немало денег за лицензирование алгоритма RSA, и стандарт бесплатной цифровой подписи прямо повлиял бы на самую суть ее коммерческих успехов. (Примечание: DSA необязательно не нарушает патенты, мы рассмотрим эту тему позднее.)

До заявления о принятии алгоритма RSADSI вело компанию против "общего модуля," который, возможно, позволит правительству подделывать подписи. Когда было объявлено, что алгоритм не использует общий модуль, критика была продолжена с других позиций [154], как с помощью писем в NIST, так и с помощью заявлений в прессе. (Четыре письма в NIST появилось в [1326]. Читая их, не забывайте, что по крайней мере два автора, Ривест и Хеллман, были финансово заинтересованы в том, чтобы DSS не был принят.)

Многие большие компании, разрабатывающие программное обеспечение, которые уже лицензировали алгоритм RSA, также выступили против DSS. В 1982 году правительство попросило предоставить ему алгоритмы с открытым ключом для выбора одного из них в качестве стандарта [537]. После этого в течение девяти лет от NIST не было никаких известий. Такие компании, как IBM, Apple, Novell, Lotus, Northern Telecom, Microsoft, DEC и Sun потратили много денег, реализуя алгоритм RSA. Они не были заинтересованы в потере инвестиций.

Всего к концу первого периода обсуждения(28 февраля 1992 года) NIST получил 109 замечаний. Рассмотрим по порядку критические замечания в адрес DSA.

1. DSA нельзя использовать для шифрования или распределения ключей.

Правильно, но стандарт и не требует наличия этих возможностей. Это стандарт подписи. NIST подготовить стандарт шифрования с открытым ключом. NIST совершает большую ошибку, оставляя американский народ без стандарта шифрования с открытым ключом. По всей вероятности предложенный стандарт цифровой подписи будет невозможно использовать для шифрования. (Но оказывается, что возможно - см. раздел 23.3.) Это не означает, что стандарт подписи бесполезен.

2. DSA был разработан NSA, и в алгоритме могут быть специальные лазейки.

Большинство первоначальных комментариев были просто параноидальными : "Отрицание NIST существующих алгоритмов без видимых причин не внушает доверия к DSS, а усиливает подозрение, что существует тайная программа, стремящаяся позволить NIST и/или NSA вскрывать национальную криптосистему с открытым ключом" [154]. Серьезный вопрос относительно безопасности DSA был задан Аржаном Ленстрой (Arjen Lenstra) и Стюартом Хабером (Stuart Haber) из Bellcore. Он будет рассмотрен ниже.

3. DSA медленнее RSA [800].

Более или менее справедливо. Скорости генерации подписи примерно одинаковы, но проверка подписи с помощью DSA от 10 до 40 раз медленнее. Однако генерация ключей быстрее. Но эта операция неинтересна, пользователь редко применяет ее. С другой стороны проверка подписи - это наиболее частая операция.

Проблема критики в том, что существует много способов поиграть параметрами тестирования, добиваясь нужных результатов. Предварительные вычисления могут ускорить генерацию подписи DSA, но они не всегда возможны. Сторонники RSA подбирают числа так, чтобы выделить преимущества своего алгоритма, а сторонники DSA используют свой способ оптимизации. В любом случае компьютеры становятся все быстрее и быстрее. Хотя разница в скорости и существует, в большинстве приложений она не будет заметна.

4. RSA - это стандарт *de facto*.

Вот два примера подобных жалоб. Письмо Роберта Фоллета (Robert Follett), директора программы стандартизации компании IBM [570]:

> IBM считает, что NIST предложил стандарт схемы цифровой подписи, отличающийся от принимаемых международных стандартов. Пользователи и организации пользователей убедили нас в том, что поддержка международных стандартов, и использующих RSA, в самом ближайшем будущем станет необходимым условием продажи средств обеспечения безопасности.

Письмо Леса Шроера (Les Shroyer), вице-президента и директора компании Motorola [1444]:

> У нас должен быть единый, надежный, признанный всеми алгоритм цифровой подписи, который можно использовать по всему миру как между американскими и неамериканскими объектами, так и между системами компании Motorola и системами других производителей. Отсутствие других жизнеспособных технологий цифровой подписи за последние восемь лет сделало RSA фактическим стандартом. . . . Motorola и многие другие компании. . . вложили в RSA миллионы долларов. Мы сомневаемся во взаимодействии и возможности поддержки двух различных стандартов , такое положение приведет к росту расходов, задержек развертывания и усложнению систем. . . .

Многим компаниям хотелось, чтобы NIST принял ISO 9796, международный стандарт цифровой подписи, использующий RSA [762.]. Хотя это и серьезный аргумент, он недостаточен, чтобы принять международный стандарт в качестве национального. Бесплатный стандарт лучше отвечал бы общественным интересам Соединенных Штатов.

5. Выбор национального алгоритма не был открытым, не было дано достаточно времени для анализа .

Сначала NIST утверждал, что разработал DSA самостоятельно, затем признал помощь NSA. Наконец NIST подтвердил, что NSA является автором алгоритма. Это многих обеспокоило - NSA не внушает людям доверие. Даже так, алгоритм был опубликован и доступен для анализа, кроме того, NIST продлил время анализа и комментирования алгоритма.

6. DSA может нарушать другие патенты. Это так. Этот вопрос будет рассмотрен в разделе, рассматривающим патенты.

7. Размер ключа слишком мал.

Это единственно справедливая критика DSS. Первоначально предлагалось использовать модуль длиной 512 битов [1149]. Так как безопасность алгоритма определяется сложностью вычисления дискретных логарифмов по заданному модулю, этот вопрос волновал многих криптографов. С тех пор вычисление дискретных логарифмов в конечном поле достигло определенных успехов, и 512 битов слишком мало для долговременной подписи (см. раздел 7.2). Согласно Брайану ЛаМаччиа (Brian LaMacchia) и Эндрю Одлыжко (Andrew Odlyzko), " . . . даже безопасность, обеспечиваемая 512-битовыми простыми числами, по видимому, находится на пределе . . . " [934]. В ответ на эти замечания NIST сделал длину ключа переменной, от 512 до 1024 битов. Немного, но все-таки получше.

19 мая 1994 года был издан окончательный вариант стандарта [1154]. При этом было сказано [542]:

> Этот стандарт может применяться всеми Федеральными департаментами и управлениями для защиты несекретной информации. . . . Этот стандарт будет использован при проектировании и реализации схем подписи с открытыми ключами, которые разрабатывают Федеральные департаменты и управления, или которые разрабатываются по из заказу . Частные и коммерческие организации могут принять и использовать этот стандарт.

Прежде чем пользоваться этим стандартом и реализовывать его, прочтите ниже раздел о патентах.

Описание DSA

DSA, представляющий собой вариант алгоритмов подписи Schnorr и EIGamal, полностью описан в [1154].

Алгоритм использует следующие параметры:

p = простое число длиной L битов, где L принимает значение, кратное 64, в диапазоне от 512 до 1024. (В первоначальном стандарте размер p был фиксирован и равен 512 битам [1149]. Это вызвало множество критических замечаний, и NIST этот пункт алгоритма [1154].)

q = 160-битовой простое число - множитель p-1.

$g = h^{(p-1)/q} \bmod p$, где h - любое число, меньшее p-1, для которого $h^{(p-1)/q} \bmod p$ больше 1.

x = число, меньшее q.

$y = g^x \bmod p$.

В алгоритме также используется однонаправленная хэш-функция: $H(m)$. Стандарт определяет использование SHA, рассмотренного в разделе 18.7.

Первые три параметра, p, q и g, открыты и могут быть общими для пользователей сети. Закрытым ключом является x, а открытым - y. Чтобы подписать сообщение, m:

(1) Алиса генерирует случайное число k, меньшее q

(2) Алиса генерирует

$r = (g^k \bmod p) \bmod q$

$s = (k^{-1} (H(m) + xr)) \bmod q$

Ее подписью служат параметры r и s, она посылает их Бобу.

(3) Боб проверяет подпись, вычисляя

$w = s^{-1} \bmod q$

$u_1 = (H(m) * w) \bmod q$

$u_2 = (rw) \bmod q$

$v = ((g^{u_1} * y^{u_2}) \bmod p) \bmod q$

Если $v = r$, то подпись правильна.

Доказательства математических соотношений можно найти в [1154]. 19th представляет собой краткое описание алгоритма.

Табл. 20-1.
Подписи DSA

	Открытый ключ:
p	простое число длиной от 512 до 1024 битов (может использоваться группой пользователей)
q	160-битовый простой множитель p-1 (может использоваться группой пользователей)
g	$= h^{(p-1)/q} \bmod p$, где h - любое число, меньшее p-1, для которого $h^{(p-1)/q} \bmod p > 1$ (может использоваться группой пользователей)
y	$= g^x \bmod p$ (p-битовое число)
	Закрытый ключ:
x	$< q$ (160-битовое число)
	Подпись:
k	выбирается случайно, меньшее q
r	(подпись) $= (g^k \bmod p) \bmod q$
s	(подпись) $= (k^{-1} (H(m) + xr)) \bmod q$
	Проверка:
w	$= s^{-1} \bmod q$
u_1	$= (H(m) * w) \bmod q$
u_2	$= (rw) \bmod q$

$v = ((g^{u_1} * y^{u_2}) \bmod p) \bmod q$

Если $v = r$, то подпись правильна.

Ускоряющие предварительные вычисления

В 18-й приведены примеры скорости работы программных реализаций DSA [918].

Табл. 20-2.
Скорость DSA для различных длин модулей с 160-битовым показателем степени (на SPARC II)

	512 битов	768 битов	1024 бита
Подпись	0.20 с	0.43 с	0.57 с
Проверка	0.35 с	0.80 с	1.27 с

Практические реализации DSA часто можно ускорить с помощью предварительных вычислений. Обратите внимание, что значение r не зависит от сообщения. Можно создать строку случайных значений k, и затем рассчитать значения r для каждого из них. Можно также вычислить k^{-1} для каждого из этих значений k. Затем, когда приходит сообщение, можно вычислить s для заданных r и k^{-1}.

Эти предварительные вычисления заметно ускоряют DSA. В 17-й приведены сравнения времени вычисления DSA и RSA для конкретной реализации интеллектуальной карточки [1479].

Табл. 20-3.
Сравнение времени вычислений RSA и DSA

	DSA	RSA	DSA с общими p, q, g
Глобальные вычисления	Off-card (P)	N/A	Off-card (P)
Генерация ключа	14 с	Off-card (S)	4с
Предварительные вычисления	14 с	N/A	4 с
Подпись	0.03 с	15 с	0.03 с
Проверка	16 с	1.5 с	10 с
	1-5 с off-card (P)	1-3 с off-card (P)	

Вычисления вне карточки (off-card) выполнялись на персональном компьютере i80386/33 МГц. (P) указывает открытые параметры off-card, а (S) - на закрытые параметры off-card. В обоих алгоритмах используется 512-битовый модуль.

Генерация простых чисел DSA

Ленстра и Хабер указали, что взломать некоторые модули намного легче, чем другие [950]. Если кто-нибудь заставит пользователей сети использовать один из таких слабых модулей, то их подписи будет легче подделать. Тем не менее это не представляет проблемы по двум причинам: такие модули легко обнаружить, и они так редки, что вероятность случайно использовать одного из них пренебрежимо мала, меньше, чем вероятность случайно получить составное число на выходе вероятностной процедуры генерации простых чисел.

В [1154] NIST рекомендовал конкретный метод генерации двух простых чисел, p и q, где q является делителем p-1. Длина простого числа p - между 512 и 1024 и кратна 64 -битам. Пусть L-1= 160n+b, где L - это длина p, а n и b - два числа, причем b меньше 160.

(1) Выберем произвольную последовательность, по крайней мере, 160 битов и назовем ее S. Пусть g - это длина S в битах.

(2) Вычислим $U = \mathrm{SHA}(S) \oplus \mathrm{SHA}((S + 1) \bmod 2^g)$, где SHA описан в разделе 18.7.

(3) Образуем q, установив наибольший и наименьший значащие биты U в 1.

(4) Проверим, является ли q простым.

(5) Если q не является простым, то вернемся на этап (1).

(6) Пусть C=0 и N=2.

(7) Для k=0,l,...,n, пусть V_k=SHA((S+N+k) mod 2^g)

(8) Пусть W - целое число

$W = V_0 + 2^{160}V_1 + \ldots + 2^{160(n-1)}\ V_{n-1} + 2^{160}\,(V_n \bmod 2^b)$

и пусть

$X = W + 2^{L-1}$

Обратите внимание, что X - это L-битовое число.

(9) Пусть $p = X - ((X \bmod 2q) - 1)$. Обратите внимание, что p конгруэнтно 1 mod $2q$.

(10) Если $p < 2^{L-1}$, то перейдем на этап (13).

(11) Проверим, является ли p простым числом.

(12) Если p - простое, перейдем к этапу (15).

(13) Пусть C=C+1 и N=N+n+l.

(14) Если C = 4096, вернемся к этапу (1). В противном случае перейдем на этап (7).

(15) Сохраним значения S и C, использованные для генерации p и q.

В [1154] переменная S называется стартовой, переменная C - счетчиком, а N - смещением.

Смысл этого упражнения в том, что оно является опубликованным способом генерации p и q. Для всех практических применений этот метод позволяет избежать слабых значений p и q. Если кто-то вручит вам какие-то p и q, вас может заинтересовать, как получены эти числа. Однако, если вы получите значения S и C, использованные при генерации случайных p и q, вы сможете повторить всю процедуру самостоятельно. Использование однонаправленной хэш-функции (в стандарте используется SHA) не позволяет получить S и C по значениям p и q.

Эта безопасность лучше, чем обеспечиваемая RSA. В RSA простые числа хранятся в секрете. Любой может генерировать фальшивое простое число или число, форма которого упрощает разложение на множители. Не зная закрытого ключа, это никогда не проверишь. В DSA, даже если закрытый ключ неизвестен, можно убедиться, что p и q генерировались случайным образом.

Шифрование ElGamal с DSA

Утверждалось, что DSA так нравится правительству, потому что его нельзя использовать в качестве алгоритма шифрования. Однако можно использовать вызов функции DSA для шифрования EIGamal. Пусть алгоритм реализован как вызов одной функции

```
DSAsign(p,q,g,k,x,h,r,s)
```

Задав входные значения p, q, g, k, x и h, можно получить параметры подписи: r и s.

Для шифрования сообщения m алгоритмом EIGamal с помощью открытого ключа y выберем случайное число k и вызовем

```
DSAsign(p,p,g,k,0,0,r,s)
```

Возвращенное значение r и будет a из схемы EIGamal. Отбросим s. Затем вызовем, call

```
DSAsign(p,p,y,k,0,0,r,s)
```

Переименуем значение r в u, отбросим s. Вызовем

```
DSAsign(p,p,m,1,u,0,r,s)
```

Отбросим r. Возвращенное значение s и будет b в схеме EIGamal. Теперь у вас есть шифротекст, a и b. Дешифрирование также просто. Используя закрытый ключ x и шифротекст сообщений, a и b, вызовем

```
DSAsign(p,p,a,x,0,0,r,s)
```

Значение r - это a^x mod p. Назовем его e. Затем вызовем

```
DSAsign(p,p,1,e,b,0,r,s)
```

Значение s и будет открытым текстом сообщения, m.

Этот способ работает не со всеми реализациями DSA - в некоторых из них могут быть зафиксированы значения p и q или длины некоторых других параметров. Тем не менее, если реализация является достаточно общей, то можно шифровать сообщение, не используя ничего, кроме функции цифровой подписи.

Шифрование RSA с DSA

Шифрование RSA еще проще. Используя модуль n, сообщение m и открытый ключ e, вызовем

```
DSAsign(n,n,m,e,0,0,r,s)
```

Возвращенное значение r и есть шифротекст. Дешифрирование RSA является точно таким же. Если d - закрытый ключ, то

```
DSAsign(n,n,m,d,0,0,r,s)
```

возвращает открытый текст как значение r.

Безопасность DSA

С 512 битами DSA недостаточно надежен для длительной безопасности, но он вполне надежен при 1024 битах. В своем первом заявлении на эту тему NSA так комментировало утверждение Джо Эбернети (Joe Abernathy) из *The Houston Chronicle* по поводу лазейки в DSS [363]:

> Что касается предполагаемой лазейки в DSS. Мы считаем, что термин "лазейка" вводит в заблуждение, так он предполагает, что через лазейку можно как-то расшифровать (прочитать) зашифрованные сообщения, подписываемые с помощью DSS, без разрешения отправителя.
>
> DSS не шифрует никаких данных. По сути вопросом является, не может ли кто-то при помощи DSS подделать подпись, и, таким образом, дискредитировать всю систему. Мы категорически заявляем, что вероятность, что кто-нибудь - включая NSA - сможет подделать подпись DSS, при правильном использовании стандарта бесконечно мала.
>
> Более того, предположение о чувствительности к лазейке справедливо для *любой* системы проверки подлинности с открытыми ключами, включая RSA. Утверждение, что это влияет только на DSS (аргумент, популярный в прессе), полностью неверно. Вопрос в реализации и способе выбора простых чисел. Мы призываем вас уделить внимание недавней конференции EUROCRYPT, где "за круглым столом" обсуждался вопрос лазеек в DSS. Одним из участников обсуждения был один из исследователей из Bellcore, утверждавший о возможности лазейки, и по нашему пониманию участники дискуссии - включая этого исследователя из Bellcore - пришли к выводу, что вопрос о лазейке в DSS не представляет проблемы. Более того, всеми было признано, что вопрос о лазейке является тривиальным и был раздут прессой. Однако, пытаясь по просьбе NIST ответить на обвинение о лазейке, мы разработали процесс генерации простых чисел, позволяющий избежать выбора одного из относительно небольшого числа слабых простых чисел, использование которых ослабляет DSS. Кроме того, NIST настаивает на использовании модулей большей длины, вплоть до 1024, что позволяет не пользоваться разработанным процессом генерации простых чисел, избегая слабых простых чисел. Очень важным дополнительным моментом, на который часто не обращают внимание, является то, что при использовании DSS простые числа *общедоступны* и, следовательно, могут быть предметом открытого изучения. Не все системы с открытыми ключами способны пройти подобную проверку.
>
> Целостность любой системы защиты информации требует обратить внимание на реализацию. Учитывая уязвимость систем с миллионами равноправных пользователей, NSA по традиции настаивает на использовании централизованных доверенных центров как на способе минимизировать риск в системе. Хотя мы по просьбе NIST и разработали ряд технических модификаций DSS, позволяющих реализовать менее централизованный подход, все же нужно выделить ту часть объявления о DSS в *Federal Register*, в которой говорится:
>
> "Хотя этот стандарт должен обеспечить общие требования безопасности генерации цифровых подписей, соответствие стандарту не обеспечивает безопасность конкретной реализации. Ответственное лицо в каждом департаменте или управлении должно гарантировать, что общая реализация гарантирует приемлемый уровень безопасности. NIST продолжит работу с правительственными пользователями, обеспечивая правильность реализаций."
>
> Наконец мы изучили все утверждения о небезопасности DSS, и они нас не убедили. DSS был тщательно изучен в NSA, что позволило нашему Директору по безопасности информационных систем разрешить использовать этот стандарт для подписи несекретных данных, обрабатываемых в определенных разведывательных системах, и даже для подписи секретных данных в ряде систем. Мы считаем, что подобное признание свидетельствует о невозможности какого-либо вероятного вскрытия безопасности, обеспечиваемой DSS при его правильных реализации и использовании. Основываясь на требованиях правительства США к технике и безопасности цифровых подписей, мы считаем, что DSS является лучшим выбором. В действительности, DSS выступает в качестве пилотного проекта Системы защиты сообщений (Defense Message System), призванного гарантировать подлинность электронных сообщений для жизненно важных команд и контрольной информации. Эта начальная демонстрация включает участие Комитета начальников штабов, военных служб и оборонных ведомств и проводится в кооперации с NIST.

Я не собираюсь комментировать истинность заявления NSA. Принимать его на вру или нет - зависит от вашего к нему отношения.

Вскрытия k

Для каждой подписи нужно новое значение k, которое должно выбираться случайным образом. Если Ева узнает k, которое Алиса использовала для подписи сообщения, может быть воспользовавшись некоторыми свойствами генератора случайных чисел, который выдает k, она сможет раскрыть закрытый ключ Алисы, x. Если Ева добудет два сообщения, подписанных с помощью одного и того же k, то, даже не зная значение k, она сможет раскрыть x. А с помощью x Ева сможет тайно подделывать подпись Алисы. В любой реализации DSA для безопасности системы очень важен хороший генератор случайных чисел [1468].

Опасности общего модуля

Хотя DSS не определяет применение пользователями общего модуля, различные реализации могут воспользоваться такой возможностью. Например, Налоговое управление рассматривает использование DSS для электронной налогов. Что если эта организация потребует, чтобы все налогоплательщики страны использовали общие p и q? Общий модуль слишком легко становится мишенью для криптоанализа. Пока слишком рано обсуждать различные реализации DSS, но причины для беспокойства есть.

Подсознательный канал в DSA

Гус Симмонс (Gus Simmons) открыл в DSA подсознательный канал [1468, 1469] (см. раздел 23.3). Этот подсознательный канал позволяет встраивать в подпись тайное сообщение, которое может быть прочитано только тем, у кого есть ключ. Согласно Симмонсу, это "замечательное совпадение", что "все очевидные недостатки подсознательных каналов, использующих схему ElGamal, могут быть устранены" в DSS, и что DSS "на сегодня обеспечивает наиболее подходящую среду для подсознательных коммуникаций". NIST и NSA не комментировали этот подсознательный канал, никто даже не знает, догадывались ли они о такой возможности. Так как этот подсознательный канал позволяет при недобросовестной реализации DSS передавать с каждой подписью часть закрытого ключа. Никогда на пользуйтесь реализацией DSS, если вы не доверяете разработчику реализации.

Патенты

Дэвид Кравиц (David Kravitz), ранее работавший в NSA, владеет патентом DSA [897]. Согласно NIST [538]:

> NIST в интересах общества стремится сделать технологию DSS доступной бесплатно по всему миру. Мы считаем, что эта технология может быть запатентована, и что никакие другие патенты не применимы к DSS, но мы не можем дать твердых гарантий этого до получения патента.

Несмотря на это, три владельца патентов утверждают, что DSA нарушает их патенты: Diffie-Hellman (см. раздел 2.2.1) [718], Merkle-Hellman (см. раздел 19.2.) [720] и Schnorr (см. раздел 21.3) [1398]. Патент Schnorr является источником наибольших сложностей. Срок действия двух других патентов истекает 1997 году, а патент Schnorr действителен до 2008 года. Алгоритм Schnorr был разработан не на правительственные деньги. В отличие от патентов PKP у правительства США нет прав на патент Schnorr, и Шнорр запатентовал свой алгоритм по всему миру. Даже если суды США вынесут решение в пользу DSA, неясно, какое решение примут суды в других странах. Сможет ли международная корпорация принять стандарт, который законен в одних странах и нарушает патентное законодательство в других? Нужно время, чтобы решить эту проблему, к моменту написания этой книги этот вопрос не решен даже в Соединенных Штатах.

В июне 1993 года NIST предложил выдать PKP исключительную патентную лицензию на DSA [541]. Соглашение провалилось после протестов общественности и стандарт вышел в свет без всяких соглашений. NIST заявил [542]:

> • . . NIST рассмотрел заявления о возможном нарушении патентов и сделал вывод, что эти заявления несправедливы.

Итак стандарт официально принят, в воздухе пахнет судебными процессами, и никто не знает, что делать. NIST заявил, что он поможет защититься людям, обвиненным в нарушении патентного законодательства при использовании DSA в работе по правительственному контракту. Остальные, по видимому, должны заботиться о себе сами. Проект банковского стандарта, использующего DSA, выдвинут ANSI [60]. NIST работает над введением стандарта DSA в правительственном аппарате. Shell Oil сделала DSA своим международным стандартом. О других предложенных стандартах DSA мне неизвестно.

20.2 Варианты DSA

Этот вариант делает проще вычисления, необходимые для подписи, не заставляя вычислять k^{-1} [1135]. Все используемые параметры - такие же, как в DSA. Для подписи сообщения m Алиса генерирует два случайных числа, k и d, меньшие q. Процедура подписи выглядит так

$r = (g^k \bmod p) \bmod q$

$s = (H(m) + xr) * d \bmod q$

$t = kd \bmod q$

Боб проверяет подпись, вычисляя

$w = t/s \bmod q$

$u_1 = (H(m) * w) \bmod q$

$u_2 = (rw) \bmod q$

Если $r = ((g^{u_1} * y^{u_2}) \bmod p) \bmod q$, то подпись правильна.

Следующий вариант упрощает вычисления при проверке подписи [1040, 1629]. Все используемые параметры - такие же, как в DSA. Для подписи сообщения m Алиса генерирует случайное число k, меньшее q. Процедура подписи выглядит так

$r = (g^k \bmod p) \bmod q$

$s = k\,(H(m) + xr)^{-1} \bmod q$

Боб проверяет подпись, вычисляя

$u_1 = (H(m) * s) \bmod q$

$u_2 = (sr) \bmod q$

Если $r = ((g^{u_1} * y^{u_2}) \bmod p) \bmod q$, то подпись правильна.

Еще один вариант DSA разрешает пакетную проверку, Боб может проверять подписи пакетами [1135]. Если все подписи правильны, то работа Боба закончена. Если одна из них неправильна, то ему еще нужно понять, какая. К несчастью, это небезопасно. Либо подписывающий, либо проверяющий может легко создать набор фальшивых подписей, который удовлетворяет критерию проверки пакета подписей [974].

Существует также вариант генерации простых чисел для DSA, который включает q и используемые для генерации простых чисел параметры внутрь p. Влияет ли эта схема на безопасность DSA, все еще неизвестно.

(1) Выберем произвольную последовательность, по крайней мере, 160 битов и назовем ее S. Пусть g - это длина S в битах.

(2) Вычислим $U = \text{SHA}(S) \oplus \text{SHA}((S + 1) \bmod 2^g)$, где SHA описан в разделе 18.7.

(3) Образуем q, установив наибольший и наименьший значащие биты U в 1.

(4) Проверим, является ли q простым.

(5) Пусть p - это объединение q, S, C и SHA(S). C представляет собой 32 нулевых бита.

(6) $p=p-(p \bmod q)+1$.

(7) $p=p+q$.

(8) Если C в p равно 0x7fffffff, вернемся на этап (1).

(9) Проверим, является ли p простым.

(10) Если p - составное, вернемся на этап (7).

Преимуществом этого варианта является то, что вам не нужно хранить значения C и S, использованные для генерации p и q. Они включены в состав p. Для приложений, работающих в условиях нехватки памяти, например, для интеллектуальных карточек, это может быть важно.

20.3 Алгоритм цифровой подписи ГОСТ

Это русский стандарт цифровой подписи, Официально называемый ГОСТ Р 34.10-94 [656]. Алгоритм очень похож на DSA, и использует следующие параметры

p = простое число, длина которого либо между 509 и 512 битами, либо между 1020 и 1024 битами.

q = простое число - множитель p-1, длиной от 254 до 256 битов.

a = любое число, меньшее p-1, для которого $a^q \bmod p = 1$.

x = число, меньшее q.

$y = a^x \bmod p$.

Этот алгоритм также использует однонаправленную хэш-функцию : $H(x)$. Стандарт определяет использование хэш-функции ГОСТ Р 34.1 1-94 (см. раздел 18.1 1), основанной на симметричном алгоритме ГОСТ (см. раздел 14.1) [657].

Первые три параметра, p, q и a, открыты и могут использоваться совместно пользователями сети. Закрытым ключом служит x, а открытым - y. Чтобы подписать сообщение m

(1) Алиса генерирует случайное число k, меньшее q

(2) Алиса генерирует $I = (\text{a}* \bmod p) \bmod q$ s = (ct + $k(H(m)))$ mod q

$r = (a^k \bmod p) \bmod q$

$s = (xr + k(H(m))) \bmod q$

Если $H(m) \bmod q = 0$, то значение хэш-функции устанавливается равным 1. Если $r = 0$, то выберите другое значение k и начните снова. Подписью служат два числа: $r \bmod 2^{256}$ и $s \bmod 2^{256}$, Алиса посылает их Бобу.

(3) Боб проверяет подпись, вычисляя

$v = H(m)^{q-2} \bmod q$

$z_1 = (sv) \bmod q$

$z_2 = ((q-r)*v) \bmod q$

$u = ((a^{z_1} * y^{z_2}) \bmod p) \bmod q$

Если $u = r$, то подпись правильна.

Различие между этой схемой и DSA в том, что в DSA $s = (k^{-1} (H(m) + xr)) \bmod q$, что дает другое уравнение проверки. Любопытно, однако, что длина q равна 256 битам. Большинству западных криптографов кажется достаточным q примерно 160 битов длиной. Может быть это просто следствие русской привычки играть в сверхбезопасность.

Стандарт используется с начала 1995 года и не закрыт грифом "Для служебного пользования", что бы это не значило.

20.4 Схемы цифровой подписи с использованием дискретных логарифмов

Схемы подписи ElGamal, Schnorr (см. раздел 21.3) и DSA очень похожи. По сути, все они являются тремя примерами общей схемы цифровой подписи, использующей проблему дискретных логарифмов . Вместе с тысячами других схем подписей они являются частью одного и того же семейства [740, 741, 699, 1184].

Выберем p, большое простое число, и q, равное либо p-1, либо большому простому множителю p-1. Затем выберем g, число между 1 и p, для которого $g^q \equiv 1 \pmod p$. Все эти числа открыты, и могут быть совместно использованы группой пользователей. Закрытым ключом является x, меньшее q. Открытым ключом служит $y = g^x \bmod q$.

Чтобы подписать сообщение m, сначала выберем случайное значение k, меньшее q и взаимно простое с ним. Если q тоже простое число, то будет работать любое k, меньшее q. Сначала вычислим

$r = g^k \bmod p$

Обобщенное **уравнение подписи** примет вид

$ak = b + cx \bmod q$

Коэффициенты a, b и c могут принимать различные значения. Каждая строка 16th предоставляет шесть возможностей. Проверяя подпись, получатель должен убедиться, что

$r^a = g^b y^c \bmod p$

Это уравнение называется **уравнением проверки**.

Табл. 20-4.
Возможные перестановки *a*, *b* и *c* ($r' = r \bmod q$)

$\pm r'$	$\pm s$	m
$\pm r' m$	$\pm s$	1
$\pm r' m$	$\pm ms$	1
$\pm m r'$	$\pm r' s$	1
$\pm ms$	$\pm r' s$	1

В 15th перечислены возможные варианты подписи и проверки, полученные только из первой строки возможных значений a, b и c без учета эффектов ±.

Табл. 20-5.
Схемы цифровой подписи с использованием дискретных логарифмов

Уравнение подписи	Уравнение проверки
(1) $r'k=s+mx \bmod q$	$r^{r'}=g^s y^m \bmod p$
(2) $r'k=m+sx \bmod q$	$r^{r'}=g^m y^s \bmod p$
(3) $sk=r'+mx \bmod q$	$r^s=g^{r'} y^m \bmod p$

(4) $sk = m + r'x \bmod q$ $\quad r^s = g^m y^{r'} \bmod p$

(5) $mk = s + r'x \bmod q$ $\quad r^m = g^s y^{r'} \bmod p$

(6) $mk = r' + sx \bmod q$ $\quad r^m = g^{r'} y^s \bmod p$

Это шесть различных схем цифровых подписей. Добавление минуса увеличивает их количество до 24. При использовании всех возможных значения a, b и c число схем доходит 120.

EIGamal [518, 519] и DSA [1154] по существу основаны на уравнении (4). Другие схемы - на уравнении (2) [24, 1629]. Schnorr [1396, 1397], как и другая схема [1183], тесно связан с уравнением (5). А уравнение (1) можно изменить так, чтобы получить схему, предложенную в [1630]. Оставшиеся уравнения - новые.

Далее. Любую из этих схем можно сделать более DSA-подобной, определив r как

$r = (g^k \bmod p) \bmod q$

Используйте то же уравнение подписи и сделайте уравнением проверки

$u_1 = a^{-1} b \bmod q$

$u_2 = a^{-1} c \bmod q$

$v = ((g^{u_1} * y^{u_2}) \bmod p) \bmod q$

$(r \bmod q)^a = g^b y^c \bmod p$

Существуют и две другие возможности подобных преобразований [740, 741]. Такие операции можно проделать с каждой из 120 схем, доведя общее число схем цифровой подписи, использующих дискретные логарифмы, до 480.

Но и это еще не все. Дополнительные обобщения и изменения приводят более, чем к 13000 вариантам (не все из них достаточно эффективны) [740, 741].

Одной из приятных сторон использования RSA для цифровой подписи является свойство, называемое **восстановлением сообщения**. Когда вы проверяете подпись RSA, вы вычисляете m. Затем вычисленное m сравнивается с сообщением и проверяется, правильна ли подпись сообщения. В предыдущих схемах восстановить m при вычислении подписи невозможно, вам потребуется вероятное m, которое и используется в уравнении проверки. Но, оказывается, можно построить вариант с восстановлением сообщения для всех вышеприведенных схем. Для подписи сначала вычислим

$r = mg^k \bmod p$

и заменим m единицей в уравнении подписи. Затем можно восстановит уравнение проверки так, чтобы m могло быть вычислено непосредственно. То же самое можно предпринять для DSA-подобных схем:

$r = (mg^k \bmod p) \bmod q$

Безопасность всех вариантов одинакова, поэтому имеет смысл выбирать схему по сложности вычисления. Большинство схем замедляет необходимость вычислять обратные значения. Как оказывается, одна из этих схем позволяет вычислять и уравнение подписи, и уравнение проверки без использования обратных значений, при этом еще и восстанавливая сообщение. Она называется схемой **p-NEW** [1184].

$r = mg^{-k} \bmod p$

$s = k - r'x \bmod q$

m восстанавливается (и проверяется подпись) с помощью вычисления

$m = g^s y^{r'} r \bmod p$

В ряде вариантов одновременно подписывается по два-три блока сообщения [740], другие варианты можно использовать для слепых подписей [741].

Это значительная область для изучения. Все различные схемы цифровой подписи с использованием дискретных логарифмов были объединены логическим каркасом. Лично я считаю, что это окончательно положит конец спорам между Schnorr [1398] и DSA [897]: DSA не является ни производной Schnorr, равно как и EIGamal. Все три алгоритма являются частными случаями описанной общей схемы, и эта общая схема незапатентована.

20.5 ONG-SCHNORR-SHAMIR

Эта схема подписи использует многочлены по модулю n [1219, 1220]. Выбирается большое целое число (знать разложение n на множители не обязательно). Затем выбирается случайное число k, взаимно простое с n, и вычисляется h, равное

$h = -k^{-2} \bmod n = -(k^{-1})^2 \bmod n$

Открытым ключом служат h и n; а закрытым - k.

Чтобы подписать сообщение M, сначала генерируется случайное число r, взаимно простое с n. Затем вычисляется:

$S_1 = 1/2\ (M/r + r) \bmod n$

$S_2 = 1/2\ (M/r - r) \bmod n$

Пара чисел S_1 и S_2 представляет собой подпись. Проверяя подпись, убеждаются, что

$S_1^2 + h*S_2^2 \equiv M \pmod n$

Описанный здесь вариант схемы основан на квадратичных многочленах. При его опубликовании в [1217] за успешный криптоанализ было предложено вознаграждение в $100. Небезопасность схемы была доказана [1255, 18], но это не остановило ее авторов. Они предложили модификацию алгоритма, основанную на кубических многочленах, также оказавшуюся небезопасной [1255]. Авторы предложили версию на базе многочленов четвертой степени, но была взломана и она [524, 1255]. Вариант, решающий эти проблемы, описан в [1134].

20.6 ESIGN

ESICN -это схема цифровой подписи, разработанная NTT Japan [1205, 583]. Утверждалось, что она не менее безопасна, чем RSA или DSA, и намного быстрее при тех же размерах ключа и подписи. Закрытым ключом служит пара больших простых чисел p и q. Открытым ключом является n, для которого

$n = p^2*q$

H - это хэш-функция, применяемая к сообщению m, причем значение $H(m)$ находится в пределах от 0 до n-1. Используется также параметр безопасности k, который будет вкратце рассмотрен.

(1) Алиса выбирает случайное число x, меньшее pq.

(2) Алиса вычисляет:

w, наименьшее целое, которое больше или равно

$(H(m) - x^k \bmod n)/pq$

$s = x + ((w/kx^{k-1} \bmod p)\ pq$

(3) Алиса посылает s Бобу.

(4) Для проверки подписи Боб вычисляет $s^k \bmod n$. Кроме этого, он вычисляет a, наименьшее целое, которое больше или равно удвоенному числу битов n, деленному на 3. Если $H(m)$ меньше или равна $s^k \bmod n$, и если $s^k \bmod n$ меньше $H(m)+2^a$, то подпись считается правильной.

Выполнив ряд предварительных вычислений, этот алгоритм можно ускорить. Эти вычисления могут быть выполнены в произвольный момент времени и никак не связаны с подписываемым сообщением. Выбрав x, Алиса может разбить этап (2) на два подэтапа. Сначала.

(2a) Алиса вычисляет:

$u = x^k \bmod n$

$v = 1/(kx^{k-1}) \bmod p$

(2b) Алиса вычисляет:

w= наименьшее целое, которое больше или равно

$(H(m) - u)/pq$

$s = x + ((wv \bmod p)\ pq$

Для обычно используемых размеров чисел предварительные вычисления ускоряют процесс подписи на порядок. Почти вся трудная работа выполняется именно на стадии предварительных вычислений. Обсуждение действий модульной арифметики, выполняемых при ускорении ESIGN, можно найти в [1625, 1624]. Этот алго-

ритм можно расширить для работы с эллиптическими кривыми [1206].

Безопасность ESIGN

Когда этот алгоритм был впервые предложен, k было выбрано равным 2 [1215]. Такая схема быстро была взломана Эрни Брикеллом (Ernie Brickell) и Джоном ДеЛаурентисом [261], которые распространили свое вскрытие и на случай $k = 3$. Модифицированная версия этого алгоритма [1203] была взломана Шамиром [1204]. Вариант, предложенный в [1204], был взломан в [1553]. ESIGN - это сегодняшняя реинкарнация алгоритмов из этого семейства. Попытка вскрыть ESIGN [963] оказалась безрезультатной.

В настоящее время авторы рекомендуют использовать следующие значения k: 8, 16, 32, 64, 128, 256, 512 и 1024. Они также рекомендуют, чтобы p и q были не меньше 192 битов каждое, образуя n не менее, чем 576 битов в длину. (Я думаю, что n должно быть еще в два раза больше.) Авторы считают, что с такими значениями параметров, безопасность ESIGN равна безопасности RSA или Rabin. И выполненный ими анализ показывает, что скорость ESIGN намного выше, чем у RSA, EIGamal и DSA [582].

Патенты

ESICN запатентован в Соединенных Штатах [1208], Канаде, Англии, Франции, Германии и Италии. Любой, кто хочет получить лицензию на алгоритм, должен обратиться в Отдел интеллектуальной собственности NTT (Intellectual Property Department, NTT, 1-6Uchisaiwai-cho, 1-chome, Chiyada-ku, 100 Japan).

20.7 Клеточные автоматы

Совершенно новая идея, изученная Папуа Гуамом (Papua Guam) [665], состоит в использовании в криптосистемах с открытыми ключами клеточных автоматов. Эта система все еще слишком нова и не прошла через тщательное изучение, но предварительное исследование показало, что у нее может быть такое же криптографически слабое место, как и у других систем [562]. Тем не менее, это многообещающая область исследований. Свойством клеточных автоматов является то, что даже если они инвертируемы, невозможно вычислить предка произвольного состояния, инвертировав правило нахождения потомка. Это выглядит очень похоже на однонаправленную хэш-функцию с лазейкой.

20.8 Другие алгоритмы с открытым ключом

За эти годы было предложено и вскрыто множество других алгоритмов с открытыми ключами. Алгоритм Matsumoto-Imai [1021] был вскрыт в [450]. Алгоритм Cade был впервые предложен в 1985 году, взломан в 1986 [774], и затем доработан в том же году [286]. Помимо этих вскрытий, существуют общие вскрытия, раскладывающие многочлены над конечными полями [605]. К любому алгоритму, безопасность которого определяется композицией многочленов над конечными полями, нужно относиться со скептицизмом, если не с откровенным подозрением.

Алгоритм Yagisawa объединяет возведение в степень mod p с арифметикой mod p-1 [1623], он был взломан в [256]. Другой алгоритм с открытым ключом, Tsujii-Kurosawa-Itoh-Fujioka-Matsumoto [1548], также оказался небезопасным [948]. Небезопасной [717] была и третья система, Luccio-Mazzone [993]. Схема подписи на базе birational перестановок [1425] была взломана на следующий день после ее представления [381]. Несколько схем подписей предложил Тацуаки Окамото (Tatsuaki Okamoto): было доказано, что одна из них так же безопасна, как проблема дискретного логарифма, а другая - как проблема дискретного логарифма *и* проблема разложения на множители [1206]. Аналогичные схемы представлены в [709].

Густавус Симмонс (Gustavus Simmons) предложил использовать в качестве основы алгоритмов с открытыми ключами J-алгебру [1455, 145]. От этой идеи пришлось отказаться после изобретения эффективных методов разложения многочленов на множители [951]. Также были изучены и специальные полугруппы многочленов [1619, 962], но и это ничего не дало. Харальд Нидеррейтер (Harald Niederreiter) предложил алгоритм с открытым ключом на базе последовательностей сдвиговых регистров [1166]. Другой алгоритм использовал слова Линдона (Lyndon) [1476], а третий - prepositional исчисление [817]. Безопасность одного из недавних алгоритмов с открытыми ключами основывалась на проблеме matrix cover [82]. Тацуаки Окамото и Казуо Ота (Kazuo Ohta) провели сравнение ряда схем цифровой подписи в [1212].

Перспективы создания радикально новых и различных алгоритмов с открытыми ключами неясны. В 1988 году Уитфилд Диффи отметил, что большинство алгоритмов с открытыми ключами основаны на одной из трех трудных проблем [492, 494]:

1. Рюкзак: Дано множество уникальных чисел, найти подмножество, сумма которого равна N.
2. Дискретный логарифм: Если p - простое число, а g и M - целые, найти x, для которого выполняется $g^x \equiv M \pmod{p}$.

3. Разложение на множители: Если N - произведение двух простых чисел, то лиьо

   (a) разложить N на множители,

   (b) для заданных целых чисел M и C найти d, для которого $M^d \equiv C \pmod{N}$,

   (c) для заданных целых чисел e и C найти M, для которого $M^e \equiv C \pmod{N}$,

   (d) для заданного целого числа x определить, существует ли целое число y, для которого $x \equiv y^2 \pmod{N}$.

Согласно Диффи [492, 494], проблема дискретных логарифмов была предложена Дж. Гиллом (J. Gill), проблема разложения на множители - Кнутом , а проблема рюкзака - самим Диффи .

Эта узость математических основ криптографии с открытыми ключами немного беспокоит . Прорыв в решении либо проблемы дискретных логарифмов, либо проблемы разложения на множители сделает небезопасными целые классы алгоритмов с открытыми ключами . Диффи показал [492, 494], что подобный риск смягчается двумя факторами:

> 1. Все операции, на которые сейчас опирается криптография с открытыми ключами - умножение, возведение в степень и разложение на множители - представляют собой фундаментальные арифметические явления . Веками они были предметом интенсивного математического изучения, и рост внимания к ним, вызванный применением в криптосистемах с открытыми ключами, увеличил, а не уменьшил наше доверие.
> 2. Наша возможность выполнять большие арифметические вычисления растет равномерно и сейчас позволяет нам реализовывать системы с числами такого размера, чтобы эти системы были чувствительны только к действительно радикальным прорывам в разложении на множители, дискретных логарифмах или извлечении корней .

Как мы уже видели, не все алгоритмы с открытыми ключами, основанные на этих проблемах, безопасны . Сила любого алгоритма с открытым ключом зависит не только от вычислительной сложности проблемы, лежащей в основе алгоритма. Трудная проблема необязательно реализуется в сильном алгоритме . Ади Шамир объясняет это тремя причинами [1415]:

1. Теория сложности обычно связана с отдельными частными случаями проблемы. Криптоаналитик же часто получает большой набор статистически связанных проблем - различные шифротексты, зашифрованные одним и тем же ключом.
2. Вычислительная сложность проблемы обычно измеряется для худшего или среднего случаев . Чтобы быть эффективной в качестве способа шифрования, проблема должна быть трудной для решения по чти во всех случаях.
3. Произвольную сложную проблему необязательно можно преобразовать в криптосистему, к тому же проблема должна позволить встроить в нее лазейку, знание которой и только оно делает возможным простое решение проблемы.