

Министерство образования и науки Российской Федерации

ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
РОССИЙСКИЙ ГОСУДАРСТВЕННЫЙ ГИДРОМЕТЕОРОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ

П.П. Бескид, Т.М. Татарникова

КРИПТОГРАФИЧЕСКИЕ МЕТОДЫ ЗАЩИТЫ ИНФОРМАЦИИ

Часть 2

АЛГОРИТМЫ, МЕТОДЫ И СРЕДСТВА ОБЕСПЕЧЕНИЯ
КОНФИДЕНЦИАЛЬНОСТИ, ПОДЛИННОСТИ И
ЦЕЛОСТНОСТИ ИНФОРМАЦИИ

Учебное пособие



Санкт-Петербург
2010

Бескид П.П., Татарникова Т.М. Криптографические методы защиты информации. Часть 2. Алгоритмы, методы и средства обеспечения конфиденциальности, подлинности и целостности информации. Учебное пособие. СПб., изд. РГГМУ, 2010. – 104 с.

Рецензент: Молдовян Н.А., д-р техн. наук, проф., зав. лабораторией криптологии СПИИРАН.

В учебном пособии представлена практическая часть дисциплины «Криптографические методы защиты информации»: описание современных криптографических алгоритмов, методов и средств обеспечения подлинности информации с помощью электронной цифровой подписи и алгоритмов обеспечения целостности информации – хэш-функций.

Предназначено для подготовки специалистов по специальности 090106 – информационная безопасность телекоммуникационных систем.

Может быть полезно при выполнении курсового проектирования, а также при выполнении лабораторных работ и практических занятий.

Beskid P.P., Tatarnikova T.M. Cryptographic methods of information protection. Part 2. Algorithms, methods and means of ensuring confidentiality, authenticity and integrity of information. A manual. St. Petersburg, RSHU Publishers, 2010. – 104 pp.

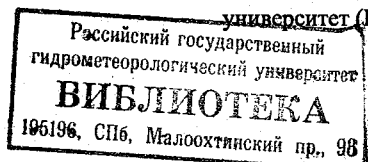
The manual presents the practical part of the discipline "Cryptographic Methods of Information Protection: description of modern cryptographic algorithms, methods and means to ensure authenticity of information through electronic signature and algorithms ensuring integrity of information - the hash functions.

Intended for training experts in Speciality 090106 'Information Security of Telecommunication Systems'.

The book may be useful for preparing course projects and doing laboratory work and practicals.

© Бескид П.П., Татарникова Т.М., 2010

© Российский государственный гидрометеорологический университет (РГГМУ), 2010



42.4. 1452

ВВЕДЕНИЕ

Криптография (cryptography) – наука о способах преобразования информации с целью защиты ее от незаконного или нежелательного использования. Одним из видов такого преобразования является шифрование информации, которое призвано обеспечивать практическую невозможность ее прочтения или изменения незаконными пользователями. Кроме того, шифрование должно также обеспечивать возможность легкого получения исходной информации из зашифрованной легитимными пользователями, которым доступен ключ алгоритма расшифрования.

Обычно предполагается, что зашифрованная информация передается по общедоступному каналу связи так, что все пользователи (законные и незаконные) имеют к ней свободный доступ.

В прикладной криптографии слова «практическая невозможность вскрытия (взламывания) шифра» означают, что незаконный пользователь будет вынужден привлекать для проведения необходимого объема вычислений или других работ по вскрытию шифра неприемлемые для него ресурсы (оборудование, время, пространство, энергию, деньги и т. п.).

В математической или теоретической криптографии стремятся показать, что задача взламывания шифра является задачей со строго доказуемыми свойствами, в частности, ее решение имеет определенную «сложность» в рамках той или иной математической модели, например, в рамках теоретико-информационного или теоретико-сложностного подхода.

В общедоступной литературе математические задачи криптографии на современном уровне впервые были рассмотрены К. Шенноном, хотя по некоторым данным в России некоторые его результаты были известны и раньше. В этой работе К. Шеннон с помощью открытого им теоретико-информационного подхода решил некоторые из важнейших проблем теоретико-информационной криптографии. В частности, им показано, что абсолютной надежностью могут обладать только те шифры, у которых объем ключа не меньше объема шифруемой информации, а также приведены примеры таких шифров. Там же были предложены и прин-

ципы построения криптографически надежных преобразований с помощью композиции некоторых разнородных отображений и т.п.

Построение стойких систем шифрования и анализ стойкости тех или иных конкретных систем шифрования являются основными задачами или проблемами практической криптографии. Под стойкостью понимается способность системы, выраженная в числе операций, противостоять попыткам ее взламывания. Под словом «взламывание» понимается определение открытой информации по известной зашифрованной информации в тех или иных условиях. Обычно усилия направляются на вычисление ключа.

Исследование стойкости проводится в тех или иных исходных предположениях о знании злоумышленником параметров системы шифрования. Наиболее часто предполагают, что злоумышленник знает открытый и зашифрованный тексты (традиционная криптографическая атака). Другим случаем является допущение возможности для злоумышленника целенаправленного подбора удобного открытого текста (атака с выбором открытого текста). Стойкость для конкретного ключа может зависеть и от класса, к которому он принадлежит, т.е. среди ключей могут встретиться и слабые. Таким образом, стойкость криптосистемы зависит от вида криптографической атаки.

Полагают, что если все рассмотренные методы не дают практически реализуемых способов восстановления ключа и не найдено других способов получения открытой информации, то система шифрования является стойкой. Естественно, что результаты анализа системы шифрования в значительной степени зависят как от способностей и квалификации исследователей, так и от уровня развития знаний в области теоретической и практической криптографии. Поэтому понятие стойкости является весьма относительным.

Многие проблемы теоретической криптографии изучаются в рамках давно сложившихся направлений математики: теории вероятностей и статистики, теории чисел, алгебры, теории кодирования, комбинаторики, теории сложности вычислений. В качестве примера укажем на методы построения рекуррентных последовательностей с определенными свойствами, методы выявления статистических закономерностей в массивах дискретной информации,

поиск эффективных способов разложения на множители больших целых чисел, свойства булевых функции и преобразований, методы дискретной оптимизации и многие другие.

Особенно большое значение для криптографии имеют результаты, связанные с построением эффективных алгоритмов решения тех или иных конкретных задач. Например, решение системы линейных уравнений, умножение матриц, вычисление значений некоторых булевых функций и преобразований, а также и многие другие являются массовыми задачами для многих методов определения ключа. Поэтому знание эффективных оценок сложности их решения позволяет делать относительно обоснованные выводы о стойкости систем шифрования.

1. КЛАССИФИКАЦИЯ ШИФРОВ ПО КЛЮЧЕВОЙ ИНФОРМАЦИИ

Одним из критериев классификации шифров является схема обработки потока информации. В соответствии с этой схемой симметричные криптоалгоритмы делятся на поточные и блочные шифры. Поточный шифр способен обрабатывать информацию побитно, т.е. подобная схема может, получив порцию из произвольного количества бит (может быть даже одного), зашифровать/дешифровать ее и передать для дальнейшей обработки другим модулям. Подобная схема удобна в каналах последовательной связи, где сам процесс передачи информации может обрываться в произвольный момент и затем через некоторый промежуток времени продолжаться дальше.

Однако побитовая обработка информации является очень медленной в тех случаях, когда вычислительная техника имеет возможности для параллельной обработки, т.е. в программной реализации. Разрядность основной массы современных процессоров равна 32 битам, существуют 64- и 80-разрядные аппаратные платформы. В этих условиях, особенно тогда, когда информация все равно переносится в буфер в том или ином объеме (пакеты в компьютерных сетях, файлы на носителях), гораздо выгоднее применять совершенно другие принципы криптографических преобразований, называемые блочными шифрами.

Основным законом блочного шифрования является «или блок, или ничего». То есть преобразования могут применяться только над информацией строго определенного объема. Размер блока на сегодняшний день равен 64, 128 или 256 битам. Частичное шифрование (скажем, попытка обработать 177 бит) невозможно. Блочное шифрование получило гораздо более широкое распространение из-за развития современной электронно-вычислительной техники, и если поточные шифры одинаково часто реализуются как программно, так и аппаратно, то блочные шифры в подавляющем большинстве имеют программную реализацию.

Асимметричные криптоалгоритмы были изначально разработаны как блочные. Теоретически возможно создание поточных асим-

метричных систем, но практическую ценность подобные схемы не имеют. Общая схема классификации шифров приведена на рис. 1.

Поточный шифратор представляет собой автомат A_k , который вырабатывает псевдослучайную последовательность $g = (g_0, \dots, g_n, \dots)$ обычно двоичную. В качестве шифрованной информации выступает последовательность $w = (w_0, \dots, w_n, \dots)$, $w_n = f(m_n, g_n)$, где $m = (m_0, \dots, m_n, \dots)$ – открытый текст. Обычно в качестве функции наложения f используется функция сложения в каком либо конечном поле или кольце, в частности, в двоичном случае $w_n = m_n \otimes g_n$. Здесь обратное преобразование (дешифрование) осуществляется точно так же: $m_n = w_n \otimes g_n$, что позволяет на обоих концах канала связи иметь шифраторы с одинаковыми ключами.



Рис. 1. Общая схема классификации шифров

Блочный шифратор представляет собой автомат A_k , входами которого является последовательность m и выходами – последовательность $w = A_k(m)$ длины n . Входная последовательность разбивается на блоки длины n и каждый блок шифруется независимо один от другого на одном ключе K . Блочный шифратор реализует перестановку элементов пространства A^n , где A – используемый алфавит. Самым известными блочными шифраторами являются американский шифратор DES (data encryption standard) и отечественный стандарт ГОСТ 28147-89, у которых длина блока n равна 64 и 256 соответственно. Имеются и другие типы шифраторов, отличные от этих, которые будут рассмотрены далее.

2. СИМЕТРИЧНАЯ КРИПТОГРАФИЯ. СТАНДАРТЫ БЛОЧНОГО ШИФРОВАНИЯ

Взлет производительности вычислительной техники внес сильные коррективы в развитие криптографии. Первыми «пострадали» примитивные операции, которые неприемлемо долго исполнялись в программной реализации. К таковым относятся, например, перестановки бит. Если в аппаратной схеме перестановка реализуется всего лишь распайкой контактов и дорожек и выполняется за один условный такт, то в универсальных программных модулях типа микропроцессора ситуация гораздо сложнее. В общем случае на перестановку или выборку каждого бита требуется одна или две команды процессора, что требует, например, для перестановки 32 бит исполнения 64 операций. Тщательная оптимизация может позволить сократить их количество за счет параллельной обработки нескольких бит в 2–3 раза, но не более того.

Следующим параметром, величина которого намного быстрее, чем ожидалось, стала неприемлемо мала, оказалась длина ключа. Экспоненциальный рост быстродействия электронно-вычислительной техники, появление параллельных архитектур и возможность быстрого объединения огромного количества ЭВМ в единую сеть свели время перебора всех вариантов 56-битного ключа до вполне приемлемых сроков. В настоящее время не взятым рубежом для стойких шифров является уже 128-битная длина ключа, во всяком случае, публично о фактах такого взлома для надежных шифров не сообщалось.

Безальтернативная тенденция к более универсальной программной реализации вынудила создателей криптоалгоритмов считаться с законами этого направления. Во-первых, для улучшения основной характеристики «скорость шифрования/стойкость» требовались не только быстрые в программном исполнении операции, но и оптимизация их разрядности под разрядность микропроцессора. За довольно продолжительным периодом 8-битных операций последовал короткий 16-битный и уступил свое место нынешней повсеместной 32-разрядной оптимизации.

Как впрочем, скоро оказалось, ориентированность под какую-либо доминирующую на данном этапе архитектуру ЭВМ не со-

всем удачное решение. Криптография все активнее входит в повседневную жизнь: электронная коммерция, электронная защита авторских прав, электронные нотариальные системы – вот неполный перечень областей ее применения. Эта тенденция приводит к тому, что программных комплексов с совершенно различными характеристиками и ресурсами, использующих криптографию, появляется все больше и больше. Хорошо построенный криптоалгоритм должен пользоваться всеми благами мощных вычислительных комплексов с огромными объемами свободной оперативной памяти и при этом не слишком медленно работать на смарт-картах с минимальными ресурсами памяти. Это заставляет исследователей искать новые универсальные операции и оптимизировать уже существующие.

Еще одной неожиданной задачей в развитии криптографических алгоритмах оказалась их оптимизация под параллельные архитектуры ЭВМ. Все чаще в специализированных системах и серверных станциях можно встретить многопроцессорные платформы. Основным ресурсом, потребляемым криптопреобразованиями, является как раз процессорное время. Количественным критерием, характеризующим способность алгоритма к распараллеливанию, является среднее количество потоков (процессоров), на которые можно разделить данную задачу. Естественно, что во многих точках алгоритма обрабатываемые данные должны сходиться вместе на одном процессоре для перемешивания, так необходимого в криптопреобразованиях. Однако количество подобных точек необходимо уменьшать в разумных пределах, а количество независимых потоков, на которые можно разделить процесс между такими точками, устремить хотя бы к четырем-шести-восьми. В целом снова получается проблема поиска золотой середины между стойкостью шифра и скоростью его работы.

2.1. Аппаратное шифрование DES

Структура DES

Традиционно первой знаменательной вехой в истории блочного шифрования считается опубликование в 1977 г. и принятие в 1980 г. в США национального Стандарта шифрования данных DES (Data Encryption Standart). Параметры алгоритма таковы: разряд-

ность блока – 64 бита, размер ключа – 56 бит. Алгоритм представляет собой классическую сеть Файстеля из 16 раундов с добавлением входной и выходной перестановок бит. Общая структура алгоритма представлена на рис. 2.

Во входной перестановке используется следующее правило перестановки бит (приведены порядковые номера бит из исходных данных, считая с «1»):

58, 50, 42, 34, 26, 18, 10, 2, 60, 52, 44, 36, 28, 20, 12, 4,
62, 54, 46, 39, 30, 22, 14, 6, 64, 56, 48, 40, 32, 24, 16, 8,
57, 49, 41, 33, 25, 17, 9, 1, 59, 51, 43, 35, 27, 19, 11, 3,
61, 53, 45, 37, 29, 21, 13, 5, 63, 55, 47, 39, 31, 23, 15, 7.

Общая формула для вычисления индексов в псевдокоде имеет вид:

цикл по i от 0 до 63

$OUT[(i_5 i_4 i_3 i_2 i_1 i_0)] = IN[(7 - (i_2 i_1 i_0)) | (i_4 i_3) | 1 - (i_5)]$,

где запись i_r обозначает r -й бит в записи числа i ,

символ $|$ – конкатенацию цифр в двоичной записи числа.

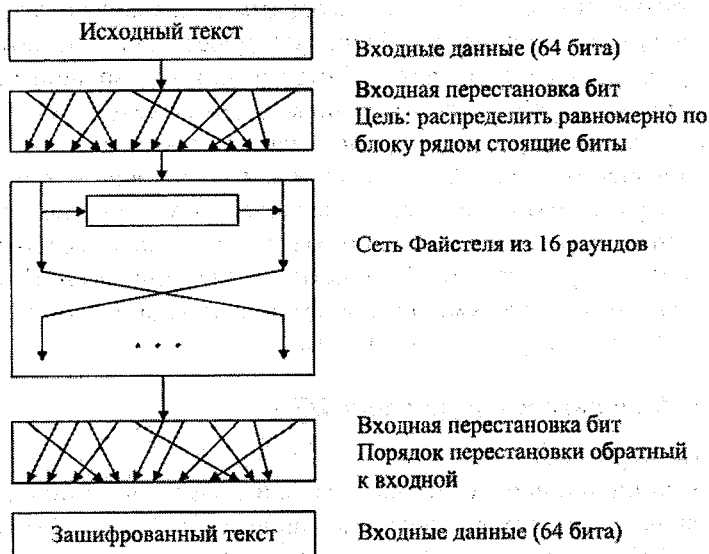


Рис. 2. Общая структура DES

Выходная перестановка бит является обратной к входной, правило перестановки бит в ней имеет вид:

40, 8, 48, 16, 56, 24, 64, 32, 39, 7, 47, 15, 55, 23, 63, 31,
 38, 6, 46, 14, 54, 22, 62, 30, 31, 5, 45, 13, 53, 21, 61, 29,
 36, 4, 44, 12, 52, 20, 60, 28, 35, 3, 3, 11, 51, 19, 59, 27,
 34, 2, 42, 10, 50, 18, 58, 26, 33, 1, 41, 9, 49, 11, 57, 25.

Добавление материала ключа производится только в сети Файстеля, поэтому, по сути, только она выполняет собственно шифрование данных.

Разрядность ключей раундов, применяемых в DES равна 48 битам. Для симметричности алгоритма шифрования/дешифрования в DES, как и в классической сети Файстеля, производится дополнительно размен последнего обмена ветвей. Это позволяет производить шифрование и дешифрование одним и тем же программным кодом, передавая ему в первом случае ключи раундов в прямом порядке ($k_0, k_1, \dots, k_{15}$), а во втором случае – в обратном ($k_{15}, k_{14}, \dots, k_0$). Общий вид сети Файстеля алгоритма DES приведен на рис. 3.

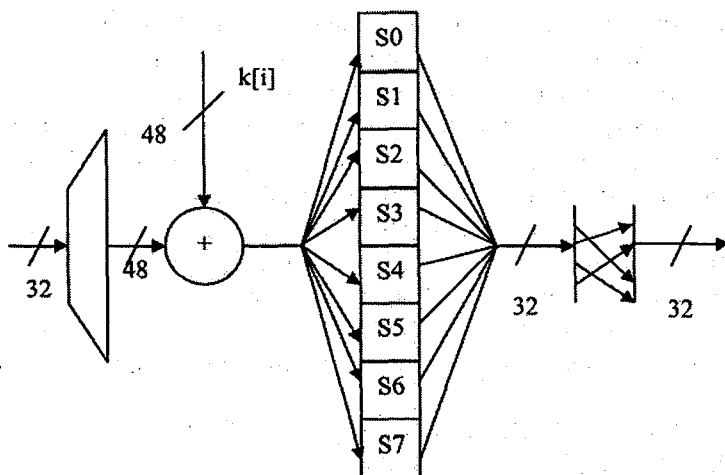


Рис. 3. Сеть Файстеля в алгоритме DES

Образующая функция сети в алгоритме DES состоит из 4 операций:

1. Перестановка бит/расширение блока с помощью повторов по определенной схеме.
2. Наложение ключа раунда операций XOR.
3. Табличные подстановки.
4. Перестановка бит.

Расширение входных данных производится на уровне бит записью некоторых из них в выходной поток дважды. В результате из 32 бит получается 48-битный блок информации, совпадающий по размеру с ключом раунда. Закон расширения выглядит следующим образом (приведены порядковые номера битов исходного блока, считая с «1»):

32, 1, 2, 3, 4, 5, 4, 5, 6, 7, 8, 9, 8, 9, 10, 11
12, 13, 12, 13, 14, 15, 16, 17, 16, 17, 18, 19, 20, 21, 20,
22, 23, 24, 25, 24, 25, 26, 27, 28, 29, 28, 29, 30, 31, 32, 1

Наложение ключа раунда производится операцией XOR, разрядность операции – 48 бит. Правило генерации ключей раунда рассмотрим позднее.

Табличные подстановки, применяемые в DES, имеют разрядность 6×4 бита, 48-битный блок данных после наложения ключа разбивается на 8 блоков по 6 бит. Каждый блок пропускается через свою табличную подстановку, обозначаемую S_0, S_1 и т.д. В результате получают 8 блоков по 4 бита в каждом: итого 32 бита – первоначальная разрядность обрабатываемой ветви.

Заключительной операцией образующей функции является перестановка бит внутри 32-битного блока, отвечающая за перемешивание информации. Правило перестановки выглядит следующим образом (приведены порядковые номера битов из 32-разрядного блока – результата табличных подстановок, считая с «1»):

16, 7, 20, 21, 29 12, 28, 17, 1, 15, 23, 26, 5, 18, 31, 10,
2, 8, 24, 14, 32, 27, 3, 9, 19, 13, 30, 6, 22, 11, 4, 25.

Общий вид одного раунда сети Файстеля алгоритма DES приведен на рис.4.

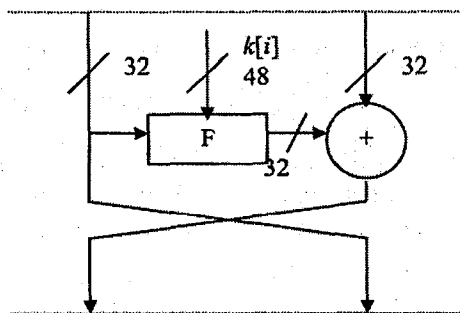


Рис. 4. Один раунд DES

DES: расширение ключа

Расширение ключа (создание ключей раунда) производится в DES по следующей схеме. На основе 56 значащих бит из ключа на начальном этапе создаются два 28-битных вектора C_0 и D_0 путем выбора бит из ключа в определенной последовательности.

Закон выбора бит из ключа для C_0 :

57, 49, 41, 33, 25, 17, 9, 1, 58, 50, 42, 34, 26, 18,
10, 2, 59, 51, 43, 35, 27, 19, 11, 3, 60, 52, 44, 36

Закон выбора бит из ключа для D_0 :

63, 55, 47, 39, 31, 23, 15, 7, 62, 54, 46, 38, 30, 22,
14, 6, 61, 53, 45, 37, 29, 21, 13, 5, 28, 20, 12, 4

Институтом стандартизации рекомендовано при хранении ключа для DES пользоваться помехоустойчивым кодированием — проверкой четности. Поэтому в каждом байте значащими являются только первые 7 бит, а восьмой добавляется при записи, исходя из правила «количество единиц в двоичной записи каждого байта нечетно». Таким образом, при истинной разрядности ключа в 56 бит хранится он в 64-битной структуре. Именно этим объясняется появление в законах выбора бит для C_0 и D_0 номеров больших 56 и отсутствие бит с номерами 8, 16, 24, 32, 40, 48, 56, 64 (они являются контрольными — незначащими).

Для генерации каждого очередного ключа раунда k_i векторы C_0 и D_0 циклически сдвигаются влево на количество разрядов, указанное в табл. 1.

Таблица 1

Величины циклических сдвигов векторов C_0 и D_0 при генерации ключа

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
ROL на ... разрядов	1	2	4	6	8	10	12	14	15	17	19	21	23	25	27	28

Для выбора 48 бит ключа i -го раунда, получившиеся после циклических сдвигов векторы $C_i D_i$ объединяются, и из 56 бит получившегося массива отбираются 48 по следующему закону:

14, 17, 11, 24, 1, 5, 3, 28, 15, 6, 21, 10, 23, 19, 12, 4,
 26, 8, 16, 7, 27, 20, 13, 2, 41, 52, 31, 37, 47, 55, 30, 40,
 51, 45, 33, 48, 44, 49, 39, 56, 34, 53, 46, 42, 50, 36, 29, 32.

Des: анализ

Первое, что бросается в глаза, – это огромное количество битовых перестановок и выборов в алгоритме. Это связано с тем, что основной схемой разработчиками планировалось только аппаратная реализация. Этому подчинено и наложение ключа именно операцией XOR, реализуемой в этом случае наиболее просто и параллельно, и небольшой размер, и минимальное количество табличных подстановок. Аппаратная реализация алгоритма на отдельной или встроенной микросхеме позволяет достичь высокой скорости шифрования при очень небольшом объеме устройства.

Единственным нелинейным элементом алгоритма является этап табличной подстановки в образующей функции Создателя алгоритма не объяснили принцип выбора используемых таблиц, однако привели несколько правил, которым удовлетворяют выбранные таблицы с целью защиты от некоторых видов криптоанализа.

Алгоритм обладает некоторыми специфическими свойствами, которые, однако, по мнению большинства специалистов, не являются значительной уязвимостью. Первым таким свойством является следующий закон:

если $Y = \text{EnCrypt}(X, KEY)$, то $Y' = \text{EnCrypt}(X', KEY')$,

где X' – побитная инверсия X , Y' – побитная инверсия Y , KEY' – побитная инверсия KEY .

Вторым свойством является наличие среди множества 2^{56} ключей четырех слабых ключей. Это следующие числа в шестнадцатирочной записи (с контрольным 8-м битом в каждом байте):

(01h 01h 01h 01h 01h 01h 01h 01h)
(Feh Feh Feh Feh Feh Feh Feh Feh)
(1Fh 1Fh 1Fh 1Fh 0Eh 0Eh 0Eh 0Eh)
(E0h E0h E0h E0h F1h F1h F1h F1h)

При шифровании данных любым из этих ключей среди 2^{64} вариантов входных данных существует 2^{32} блоков, которые в результате шифрования преобразуются сами в себя, то есть $\text{Encrypt}(X, \text{KEY}) = X$. Несмотря на то что вероятность подобного «бесполезного» шифрования очень велика ($1/2^{32}$ – это высокий риск для криптографии), вероятность того, что сам ключ окажется слабым, очень мала ($1/2^{54}$). Это позволяет считать наличие слабых ключей у DES, скорее, оригинальной особенностью, чем реальной уязвимостью.

За многие годы исследований DES зарекомендовал себя надежным шифром. Относительно него было разработано несколько криптоатак, требовавших для выяснения ключа несколько меньше операций, чем взлом полным перебором. Однако ни одна из них не имела настолько качественного превосходства над методом полного перебора, чтобы ее можно было назвать полноценным взломом алгоритма. По-настоящему заставили DES уйти с рынка конкурентно-способных блочных шифров две причины: ориентация на аппаратную реализацию и слишком малый размер ключа.

2.2. Криптоалгоритм TEA

Структура TEA

Одним из самых простых представителей семейства программно ориентированных надежных блочных шифров является TEA (Tiny Encryption Algorithm). Дословно его название переводится на русский язык как «крошечный криптоалгоритм». Разработан он был в исследовательском центре защиты информации Кембриджского университета и представляет собой классическую сеть Файстеля из 64 раундов. Схема алгоритма приведена на рис. 5. Алгоритм оптимизирован под 32-разрядные микропроцессоры, размер блока – 64 бита.

Результат образующей функции накладывается на независимую ветвь операций «сложение», поэтому восстановить исходный текст простым обращением порядка ключей раунда невозможно. Алгоритм ТЕА является примером несимметричной сети Файстеля: для дешифрования приходится реализовывать отдельную процедуру, хотя, как и шифрование, очень простую.

ТЕА: расширение ключа

Алгоритм использует 128-битный ключ, который превращается в ключи раунда по очень простой схеме. Ключ разбивается на четыре 32-разрядных машинных слова K_0, K_1, K_2, K_3 . Во всех четных (считая с 0) раундах в качестве 64-битного ключа раунда используется пара (K_0, K_1) в нечетных – пара (K_2, K_3) . В общепринятой схеме номер раунда обозначается переменной i ; раундов – r , компоненты ключа раунда – как массив $k[i]$ (он использует сквозную нумерацию по машинным словам – в нулевом раунде используются $k[0]$ и $k[1]$, в первом – $k[2]$ и $k[3]$ и т. д.). В таких обозначениях общая формула создания 128 компонент материала ключа для ТЕА выглядит следующим образом: $k[i] = K_{(i \bmod 4)}$.

Анализ ТЕА

В образующей функции ТЕА используются только битовые сдвиги, сложение и исключающее ИЛИ. Это приводит к тому, что в алгоритме полностью отсутствуют нелинейные преобразования. Стойкость к линейному криптоанализу обеспечивается большим не свойственным для сети Файстеля числом раундов. Согласно исследованиям разработчиков, ТЕА стоек ко всем опубликованным на сегодняшний день видам криптоатак. Однако из-за своей предельной простоты, которая может вызвать подозрение на наличие уязвимостей к какому-либо специфическому виду криптоанализа, алгоритм все-таки не получил широкого применения на практике.

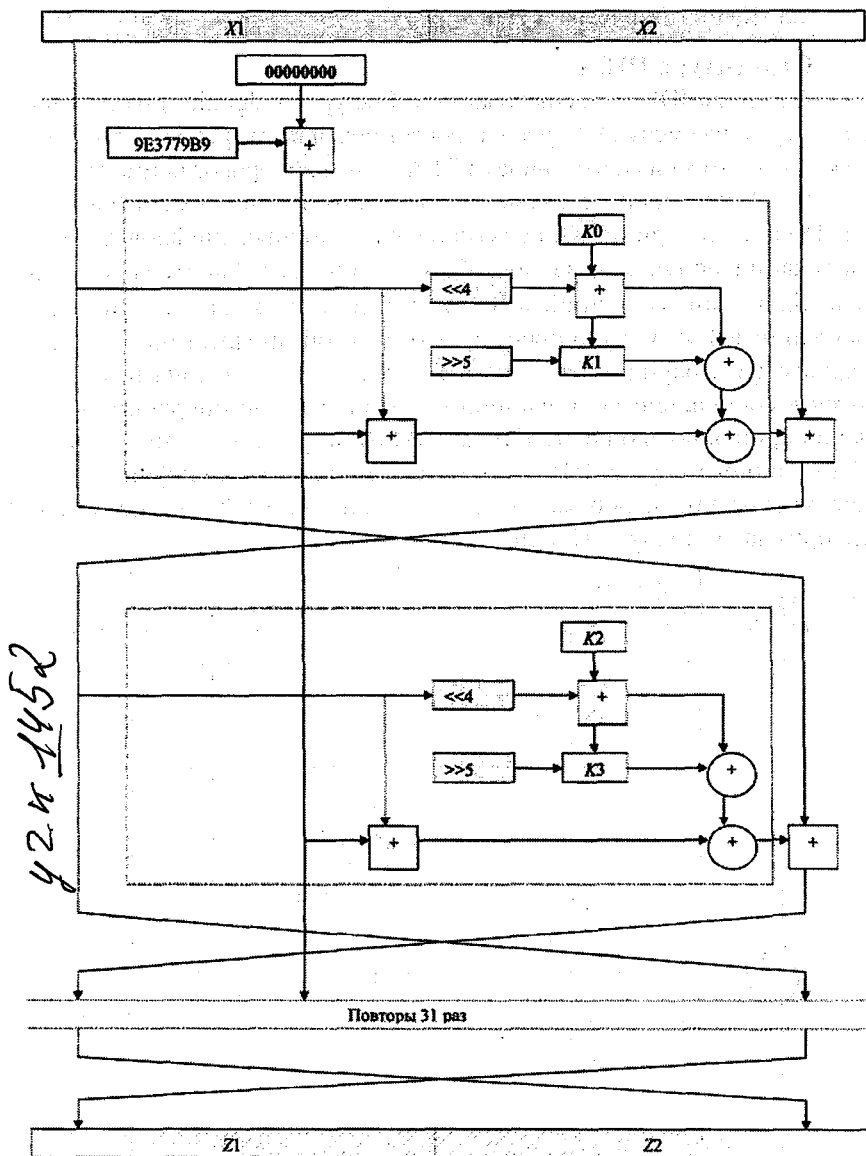


Рис. 5. Алгоритм TEA

2.3. Криптоалгоритм IDEA

Структура IDEA

Алгоритм IDEA (International Data Encryption algorithm) разработан специально на смену DES, удовлетворяет почти всем требованиям нового криптопоколения и в отличие от TEA получил широкое применение.

IDEA построен по схеме, очень напоминающей сеть Файстеля. Подобный принцип проектирования блочных шифров в классификации получил название обобщенной сети Файстеля. Исходный блок данных длиной в 64 бита разбивается на 4 ветви – все операции в IDEA выполняются в 16-битной арифметике. Затем в каждом из 8 раундов над ветвями и материалом ключа выполняются в специально подобранной очередности три операции: «сложение», «исключающее ИЛИ» и «модифицированное умножение». Схема одного раунда IDEA приведена на рис. 6. Модифицированное умножение изображено на схеме символом *. Длина ключа алгоритма составляет 128 бит.

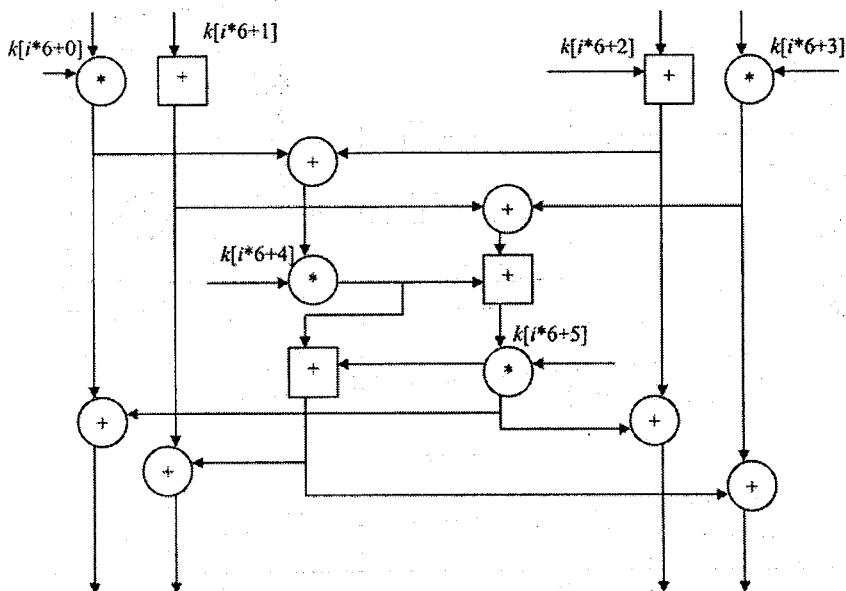


Рис. 6. Раунд алгоритма IDEA

Модифицированное умножение представляет собой обычное умножение по модулю $65537 (2^{16}+1)$ с двумя условностями. Во-первых, если какой-либо из сомножителей равен 0, то перед умножением он заменяется на 2^{16} . Во-вторых, если получившееся после взятия модуля число равно 2^{16} , то оно заменяется на 0. Определенная подобным образом операция обладает следующими свойствами:

- отображает пару 16-разрядных чисел на 16-разрядное число;
- обратима (по известному результату и одному из сомножителей возможно однозначно определить второй сомножитель).

Для достижения симметричности шифрования/дешифрования после восьми раундов над ветвями производится дополнительное преобразование. Общая схема алгоритма IDEA приведена на рис. 7. С помощью подобного дополнения создатели добились того, что для дешифрования в IDEA используется та же самая процедура, но в качестве 52 компонент материала ключа ей передается специальным образом модифицированные компоненты.

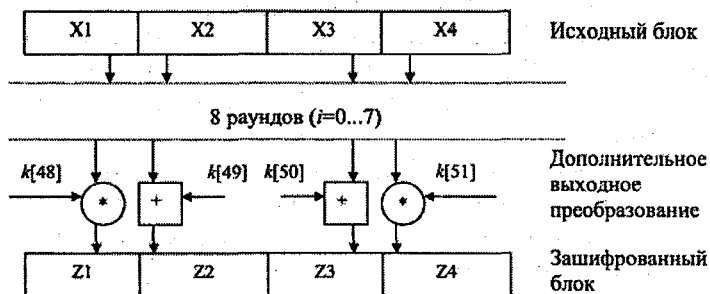


Рис. 7. Алгоритм IDEA

IDEA: расширение ключа

Получение из 128-битного ключа материала ключа в IDEA предельно просто. Первые 8 16-битных компонент $k[0]...k[7]$ материала ключа получают непосредственно разбиением 128 бит ключа на блоки без каких-либо преобразований. Затем 6 раз повторяется следующая операция: ключ сдвигается циклически влево на 25 бит и очередные 8 компонент получают таким же разбиением уже сдвинутого ключа на 16-битные блоки.

В псевдокоде расширение ключа выглядит следующим образом:

$k[0]...k[7] = \text{key}$
 $\text{key} = \text{key ROL } 25$
 $k[8]...k[15] = \text{key}$
 $\text{key} = \text{key ROL } 25$
 ...

Для дешифрования в IDEA можно использовать тот же самый основной алгоритм, но предварительно специальным образом необходимо подготовить материал ключа. Если обозначим 52 компоненты материала ключа, используемые для шифрования через $e[0]...e[51]$, а материал для дешифрования $d[0]...d[51]$, то алгоритм создания вектора d по вектору e будет иметь следующий вид:

Если $(i \bmod 6 = 0)$, то $d[i] = \text{inv}(e[48-i])$.

Если $(i \bmod 6 = 3)$, то $d[i] = \text{inv}(e[54-i])$.

Если $(i \bmod 6 = 4)$, то $d[i] = \text{inv}(e[50-i])$.

Если $(i \bmod 6 = 5)$, то $d[i] = \text{inv}(e[52-i])$.

Если $(i \bmod 6 = 1 \text{ или } 2)$, то $d[i] = -e[51-i]$, за исключением:

$d[1] = -e[49]$, $d[2] = -e[50]$, $d[49] = -e[1]$, $d[50] = -e[2]$.

Операция $\text{inv}(x)$ представляет собой вычисление числа, обратного к x относительно операции модифицированного умножения, т.е. если некоторое число a будет сначала умножено на x , а затем на $\text{inv}(x)$, то в результате всегда получится снова число a . Операция умножения по модулю $(2^{16} + 1)$ была специально модифицирована авторами с тем, чтобы такое число $\text{inv}(x)$ существовало для любого x , т.е. операция умножения была обратима. Для нахождения $\text{inv}(x)$ на практике используется алгоритм Евклида.

Как видно, основная идея инвертирования материала ключа проста. Во-первых все компоненты записываются в обратном порядке. Во-вторых, значения заменяются на такие числа, которые при повторном применении той же операции восстановят исходное значение ветви. Для арифметического сложения – это то же число, но со знаком минус, для умножения – величина $\text{inv}(x)$, а для операции XOR – само же наложенное при шифровании значение.

IDEA: анализ

Алгоритм интересен тем, что использует только три легко реализуемые на современных ЭВМ операции, не содержит таблиц

подстановок и каких-либо констант. Все три операции являются обратимыми и реализуют различные схемы «перемешивания» данных с материалом ключа.

За полтора десятка лет с момента опубликования алгоритма наиболее серьезной найденной проблемой в его криптостойкости считается наличие набора из 2^{51} слабых ключей. При случайно попадании ключа на одно из таких значений злоумышленник может при определенных условиях восстановить значение ключа за 2^{17} операций.

В целом, IDEA считается одним из очень надежных и простых блочных криптоалгоритмов, но не удовлетворяет требованиям современной вычислительной техники из-за ориентированности на 16-битную архитектуру и, как следствие, из-за более низкого быстродействия, чем могло бы быть достигнуто 32-разрядным алгоритмом при «примерно той же» криптостойкости.

2.4. Криптоалгоритм ГОСТа 28147-89

Структура ГОСТа 28147-89

Криптоалгоритм ГОСТ 28147-89 [ГОСТ 28 147-89] является легендарным и, пожалуй, единственным из официально опубликованных советских блочных шифров. Он разработан на заре блочного шифрования в недрах спецслужб Советского Союза. Постепенно шифр был рассекречен. Было ли причиной тому раскрытие его иностранными разведками или на смену ему повсеместно пришел другой более современный алгоритм, широкой общественности неизвестно. Как бы то ни было, в 1989 г. шифр был принят государственным стандартом блочного шифрования и к нему получили доступ многие специалисты по защите информации. В течение долгого времени данный алгоритм был единственной крипторазработкой, удостоившийся «ГОСТирования». Вследствие этого за ним закрепилось «безциферное» имя «ГОСТ», под которым он известен и за рубежом (GOST). Алгоритм не запатентован, что делает его привлекательным для применения в коммерческих приложениях (в том числе за границей).

ГОСТ 28147-89 оперирует 64-битным блоком данных, шифруя его с помощью 256-битного ключа. Структура криптоалгоритма предельно проста: шифр реализует классическую сеть Файстеля

из 32 раундов с двумя ветвями (рис.8). Образующая функция алгоритма приведена на рис. 9.

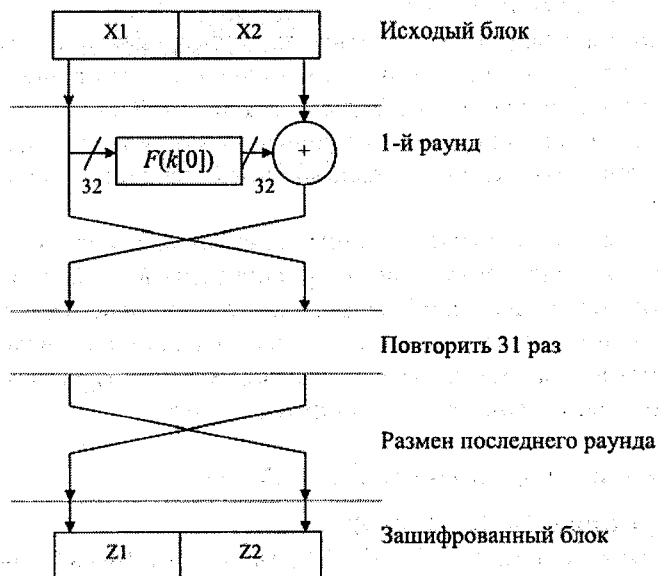


Рис. 8. Общий вид ГОСТа 28147-89

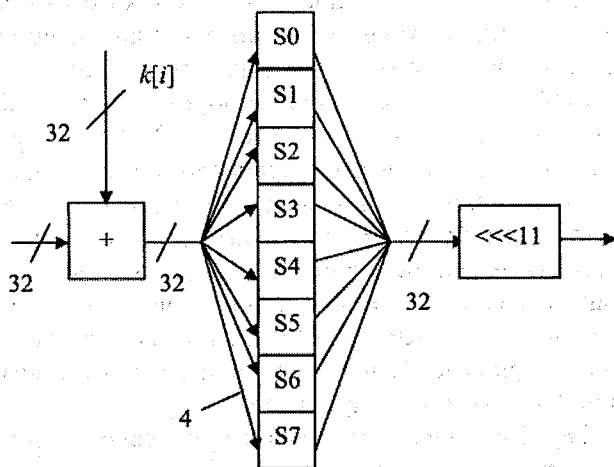


Рис. 9. Образующая функция ГОСТа 28147-89

Определив алгоритм подобным образом, авторы добились очень высокой потенциальной гибкости шифра и структуры ключей. Проблема возникла тогда, когда использовать шифр захотели многочисленные невоенные и даже негосударственные структуры. В тексте ГОСТа 28147-89 табличные подстановки не зафиксированы, более того, не приведено даже возможного их примера. Следовательно, разработчики встают перед вопросом, где взять эти значения. Очевидно, что именно табличные подстановки вносят основной вклад в стойкость алгоритма: например, при наличии в них сильной линейной корреляции шифр становится совершенно беззащитным перед линейным криптоанализом – других нелинейных криптопримитивов, кроме табличных подстановок, в его структуре просто нет.

Единственным официальным учреждением, имеющим полномочия на создание и выдачу материала табличных подстановок является ФАПСИ РФ. По-видимому, только этой организации доступен полный список прав для создания «стойких», «умеренно стойких» и «слабых» таблиц подстановок. Никаких публичных заявлений относительно правил построения таблиц официальными лицами сделано не было. Все это формирует очень специфическую ситуацию вокруг практического применения данного шифра в негосударственных структурах.

ГОСТ 28147-89: расширение ключа

Расширение ключа производится по очень простой схеме: 256-битный ключ разбивается на восемь 32-битных машинных слова. Они нумеруются с K_0 по K_7 . 32 компоненты материала ключа получают их применением в следующем порядке:

$$k[0]...k[32] = K_0 K_1... K_6 K_7 K_0 K_1... K_6 K_7 K_0 K_1... K_6 K_7 K_7 K_6... K_1 K_0.$$

В последней четвертой секции 8 машинных слов ключа записываются в обратном порядке. Для дешифрования (по свойству симметричной сети Файстеля) используется тот же код, но с обратным порядком ключей раунда:

$$K_0 K_1... K_6 K_7 K_7 K_6... K_1 K_0 K_7 K_6... K_1 K_0 K_7 K_6... K_1 K_0$$

ГОСТ 28147-89: анализ

Криптостойкость данного алгоритма зависит на 95% от таблиц подстановок. Несомненно, что существуют наборы таблиц, при использовании которых ГОСТ 28147-89 на несколько порядков устойчивее к криптоатакам, нежели DES и многие другие современные алгоритмы. Для этого у алгоритма есть все предпосылки: потенциальная емкость ключа в 256 бит, зарекомендовавшая себя структура сети Файстеля, взятое «с запасом» число раундов, не вызывающая нареканий образующая функция. Алгоритм оптимизирован под 32-разрядные процессоры и очень прост в реализации.

2.5. Алгоритмы конкурса AES

Цели и условия конкурса

Недостаточная длина ключа DES, его ориентированность на аппаратную реализацию и, как следствие, использование на практике огромного числа малоизвестных криптоалгоритмов побудило правительство Соединенных Штатов к объявлению открытого конкурса на криптостандарт блочного шифрования США начала нового века. Конкурс получил название AES (Advanced Encrypton Standart) – «улучшенный стандарт шифрования» и был проведен под эгидой Национального Института Стандартизации и Техники (National Institute of Standart – NIST) в 1997-2000 годах. Лучшие претенденты на звание AES, несомненно, отражают современное состояние дел в открытом блочном шифровании и достойны изучения.

На первом этапе конкурса заявить о своей разработке могла любая группа исследователей или компания из любой страны мира. Требования к новому стандарту были чрезвычайно просты:

- шифр должен быть блочным;
- шифр должен иметь длину блока, равную 128 битам;
- шифр должен поддерживать ключи длиной 128, 192, 256 бит.

К середине 1998 г. свои разработки на конкурс представили 15 исследовательских групп. К сожалению, российские криптографы не пожелали участвовать в этом не имеющем аналогов проекте. В течение года организаторы проекта вели переписку с научными центрами всего мира – высказываться «за» или «против» того или иного претендента имел любой желающий. Все письма (как и сами алгоритмы-конкурсанты) сразу же публиковались на сервере

NIST. В итоге за год было получено более ста электронных и печатных посланий как от научных центров с мировой известностью, так и от исследователей-специалистов в данной области.

Во второй половине 1999 г. NIST представил общественности документ, в котором сравнивались все 15 претендентов и были объявлены 5 финалистов, прошедших во второй этап конкурса. Все критерии, по которым производилось сравнение алгоритмов, были разбиты сотрудниками Института на категории: криптографические и практические характеристики.

Среди наиболее важных криптографических характеристик специалисты NIST указывали:

- стойкость к известным на момент проведения конкурса криптографическим атакам;
- максимально возможное упрощение алгоритма (в большинстве случаев – уменьшение количества раундов), при котором шифр все еще остается стойким к известным криптоатакам;
- отсутствие слабых ключей;
- стойкость к криптоатакам, специфичным для применяемых в шифре схем и криптоопераций;
- отсутствие корреляций между данными или ключом и временем, требуемым на операции шифрования/дешифрования;
- отсутствие корреляций между данными или ключом и мощностью, потребляемой аппаратным или программным устройством, реализующим шифр;
- отсутствие необоснованно сложных преобразований и «необъяснимых» констант (что дает некоторый шанс исключить недокументированные свойства шифра, известные только разработчикам);
- применение в структуре алгоритма традиционных криптосхем (парадигм) и частей уже опубликованных ранее алгоритмов, позволяющее принять во внимание результаты анализа этих схем за гораздо более длительный промежуток, чем год проведения конкурса.

К наиболее важным практическим аспектам жюри отнесло:

- оптимизацию по скорости выполнения кода (в первую очередь на 32-, 8- и 16-разрядных архитектурах ЭВМ);

- активное использование для ускорения вычислений ресурсов оперативной памяти, если таковые имеются в наличии на исполняющем программу компьютере;
- возможность оптимизации по размеру кода (для хранения в ПЗУ «слабых» микросхем и в других запоминающих устройствах с ограниченными ресурсами);
- возможность распараллеливания кода (естественно без потерь в криптостойкости);
- идентичность кода и времени исполнения процедур шифрования и дешифрования;
- сложность и затраты времени на процедуру создания материала ключа;
- характер зависимости скорости работы алгоритма от размера используемого ключа (128, 192 ади 256 бит).

Алгоритмы пяти участников, прошедших в финал, рассмотрим подробнее.

2.5.1. MARS

MARS: структура

Алгоритм MARS разработан специалистами фирмы IBM. Он представляет собой блочный шифр с размером блока, равным 128 битам, и переменной длиной ключа. Схема расширения материала ключа такова, что в качестве него может использоваться последовательность бит произвольной длины от 128 до 1248 бит. Сами авторы отмечают, что такая большая возможная длина ключа выбрана ими не с точки зрения стойкости шифра (очевидно, что барьер в 256 бит будет еще очень долго оставаться недостижимой величиной), а ради удобства использования шифра совместно с другими криптографическими примитивами. Так, например, во многих системах распространения ключей уникальная ключевая информация, получаемая в результате выполнения всех положенных действий, имеет длину в 512, 768 или 1024 бита.

Алгоритм MARS можно классифицировать как среднего уровня сложности и реализации. Он представляет собой традиционную сеть Файстеля 3-го типа с 4-мя ветвями. Однако для повышения стойкости шифра к известным и, возможно, новым схемам криптоанализа разработчики значительно усложнили дополни-

тельные входное и выходное преобразования сети Файстеля. В результате весь процесс шифрования удобно представить в виде 6 последовательных преобразований.

1. Входное сложение с материалом ключа (первичное «забеливание»).
2. Входное перемешивание данных (8 раундов преобразования E_TRANS).
3. Первые 8 раундов сети Файстеля (первая схема наложения).
4. Завершающие 8 раундов сети Файстеля (вторая схема наложения).
5. Выходное перемешивание данных (8 раундов преобразования D_TRANS).
6. Выходное сложение с материалом ключа (выходное «забеливание»).

Все шесть преобразований спроектированы авторами практически симметрично. За исключением двух центральных этапов, исполнение шифра является почти зеркальным. Подобная структура продиктована желанием защитить шифр от криптоатак абсолютно симметрично как «со стороны» входных данных, так и «со стороны» выходных данных.

Схемы входного сложения с материалом ключа выходного «забеливания» приведены соответственно на рис. 10 и 11.

Забеливание – это процедура сложения по модулю два (XOR) данных с ключами (материалом ключа) перед первым раундом и после последнего раунда. Впервые эта техника была использована Рональдом Ривестом в алгоритме шифрования DES, которое показало, что забеливание действительно усложняет задачу поиска ключа полным перебором.

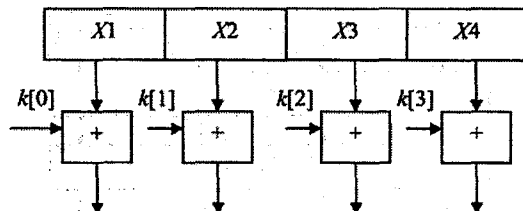


Рис. 10. Входное сложение алгоритма MARS

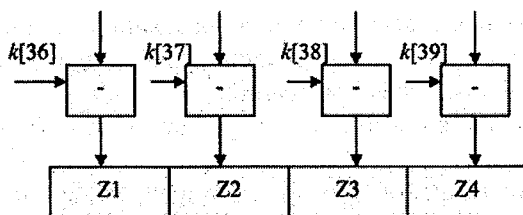


Рис. 11. Выходное «забивание» алгоритма MARS

Схемы входного и выходного перемешивания приведены на рис. 12 и 13 соответственно.

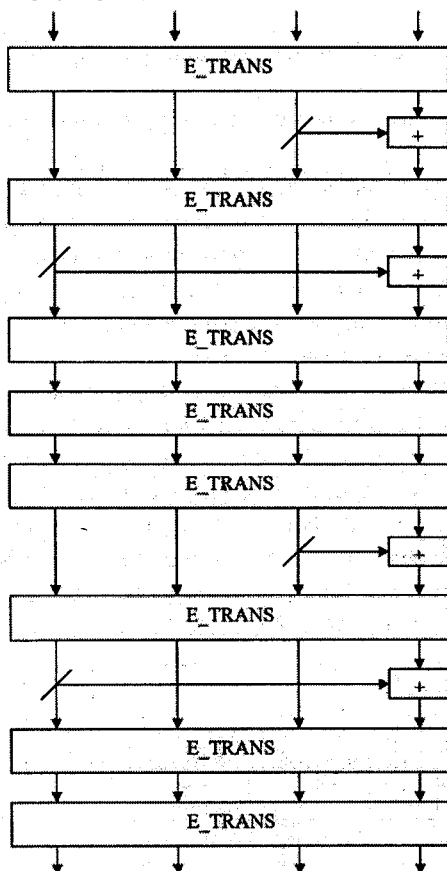


Рис. 12. Входное перемешивание MARS

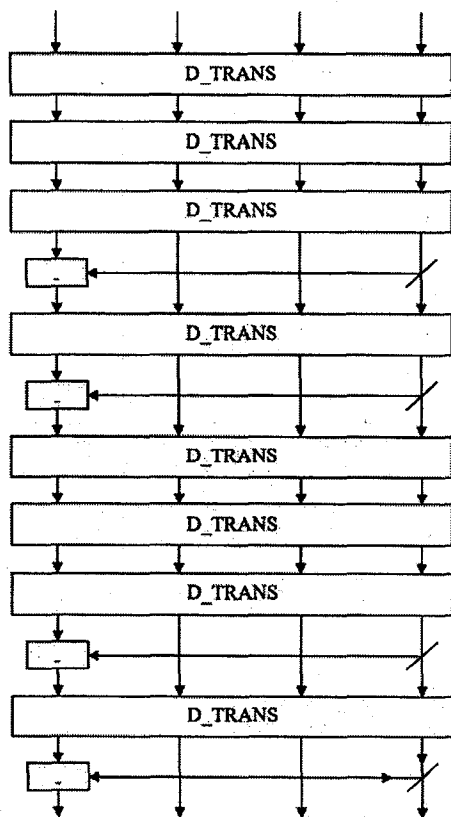


Рис. 13. Выходное перемешивание MARS

Обе процедуры состоят из 8-кратного повторения типовых схем E_TRANS и D_TRANS приведенных на рис. 14 и 15 соответственно. Разнообразят данное восьмикратное повторение только операции арифметического сложения (в обратном проходе – вычитания), выполняемые после 1, 2, 5 и 6 применений схем перемешивания. Подобная «нестройность» алгоритма лежит в полном соответствии с общей концепцией авторов сделать шифр легко анализируемым и понятным, но не приводимым к простому аналитическому виду.

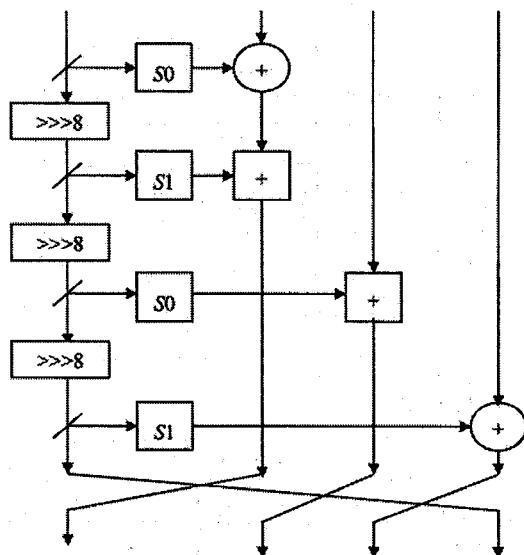


Рис. 14. Преобразование E_TRANS алгоритма MARS

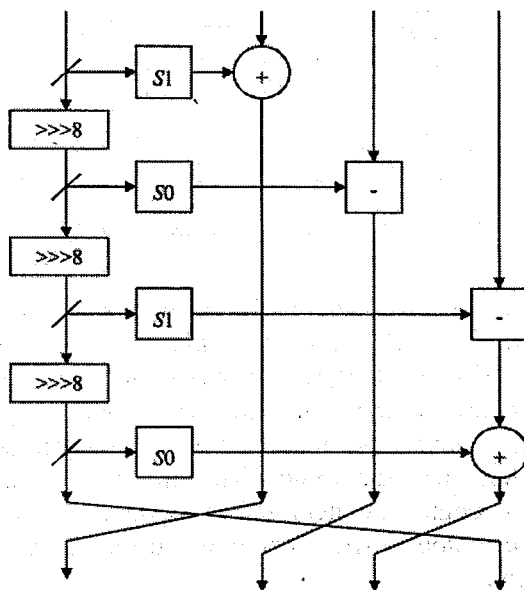


Рис. 15. Преобразование D_TRANS алгоритма MARS

Как видим, на данном этапе полностью отсутствует взаимодействие с материалом ключа. Главная цель перемешивания: подготовка информации к основному циклу криптопреобразований, улучшение статистических характеристик входного блока. Даже в том случае, если на вход криптоалгоритма подаются сильно коррелированные между собой данные, например, ASCII-текст, входное перемешивание (с учетом уже добавленной первой порции материала ключа) выравнивает все статистические характеристики блока. Кроме того, разработчики полагают, что использование в шифре разнородных этапов (перемешивание плюс сеть Файстеля) повышает потенциальную стойкость алгоритма к еще не открытым методам криптоанализа.

Табличные подстановки S_0 и S_1 на этапе перемешивания имеют разрядность 8×32 бита (т.е. в качестве индекса в табличной выборке используются только младшие 8 бит из каждой ветви). Эти же две таблицы, но объединенные вместе, образуют табличную подстановку S разрядностью 9×32 бита, используемую в основных раундах сети Файстеля.

Табличная подстановка S из 512 машинных слов каждое разрядностью в 32 бита, создана авторами по следующей схеме:

цикл по i от 0 до 102

цикл по j от 0 до 4

$S[5*i+j] = (j\text{-е машинное слово от SHA-1}(i|c_1|c_2|c_3))$

То есть для создания каждого значения в таблице подстановок запускается стандартизованная хэш-функция SHA-1 от блока данных, представляющего собой соединение счетчика цикла i и трех констант c_1, c_2, c_3 . Из получившегося 160-битного результата хэш-функции берутся только 32 бита, соответствующие состоянию счетчика цикла j .

Выбор констант c_1, c_2, c_3 авторы описывают следующим образом $c_1 = B7E15162_{16}$ (старшие 32 бита двоичной записи дробной части числа $e = 2,71828182845\dots$), $c_2 = 243F6A88_{16}$ (старшие 32 бита двоичной записи дробной части числа $\pi = 3,14159265359\dots$). Константа $c_3 = 02917B59_{16}$ была подобрана простым перебором до тех пор, пока получающаяся табличная подстановка не стала отвечать набору требований, гарантирующих стойкость к линейному и дифференциальному криптоанализу. Подобная схема создания

таблиц подстановок (с использованием широко известных констант и небольшой подбираемой части) получила в последнее время широкое распространение. Причина в том, что при достаточно хорошей рандомизации и гарантированной стойкости таблиц она позволяет убедить пользователей в отсутствии каких-либо недокументированных возможностей шифра.

Образующая функция F , используемая в алгоритме во всех 16 раундах сети Файстеля приведена на рис. 16. Как видно, в ней разработчики использовали практически все применяемые на сегодняшний день криптопримитивы: «арифметическое сложение», «исключающее ИЛИ», «32-разрядное умножение», «сдвиги» на фиксированное и переменное число бит, «табличные подстановки». Подобное разнообразие авторы объясняют повышением потенциальной стойкости алгоритма к еще неизвестным методам криптоатак: вероятность того, что какая-либо закономерность потока данных, используемая в подобной атаке, пройдет через все виды криптопреобразований, очень мала. Подобные взгляды составляют целую парадигму в современной криптографии в оппозицию принципам «чистых» математических преобразований (которые используют очень ограниченный набор операций с тщательно выбранными свойствами).

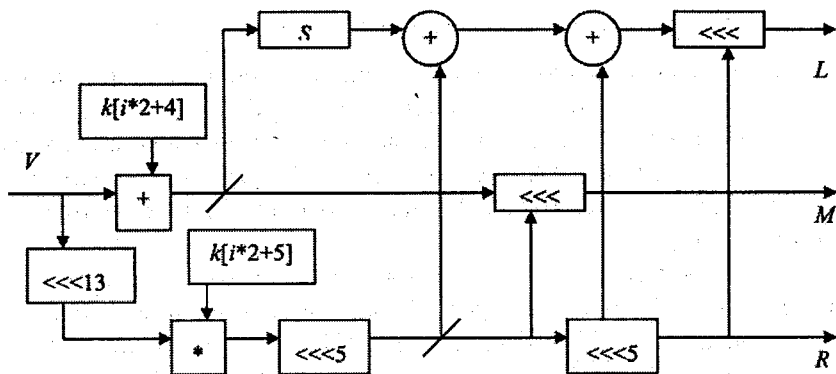


Рис. 16. Образующая функция MARS

Схема наложения трех результатов образующей функции на остальные ветви отличается в первых восьми (рис. 17) и во вторых восьми (рис. 18) раундах сети Файстеля.

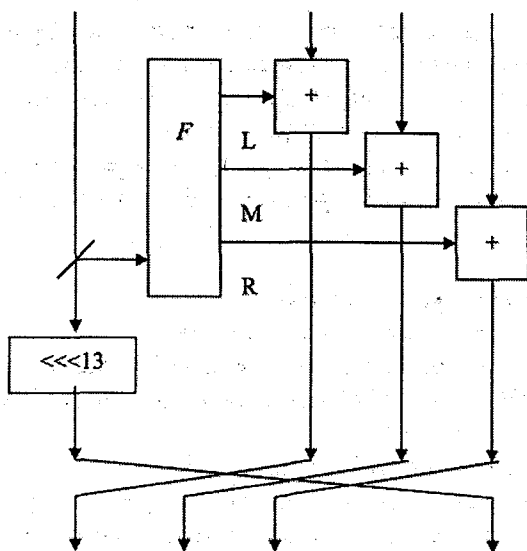


Рис. 17. Первая схема наложения MARS

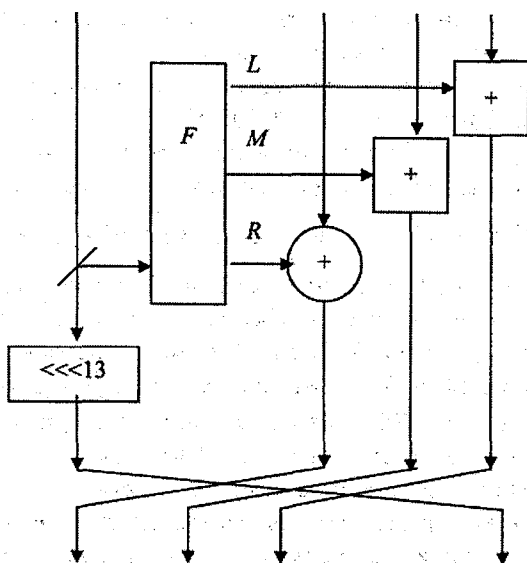


Рис. 18. вторая схема наложения MARS

Процедура дешифрования отличается от шифрования только заменой арифметического сложения на вычитание в схеме наложения результата образующей функции F на остальные ветви блока и инверсным направлением размена ветвей. Это позволяет при процедурной реализации шифра уместить обратный алгоритм (при наличии готовой «прямой» реализации) всего в несколько десятков байт кода.

MARS: Расширение ключа

Алгоритм обладает очень сложной процедурой расширения материала ключа и состоит из следующих 4-х этапов:

1. Первоначальное расширение ключа произвольной длины до 1280 бит.
2. Перемешивание материала ключа.
3. Изменение порядка ключевого материала (пересортировка).
4. Проверка и коррекция материала ключа, используемого в операции 32- битного умножения.

Первый этап расширения создает из ключа K произвольной длины n (измеряется в 32-битных машинных словах) временный массив T из 47 машинных слов по следующей схеме. Массив T имеет нумерацию элементов с -7 до $+39$.

Элементы с -7 по -1 заполняются значениями из табличной подстановки S : $T[-7] = S[0]$, ..., $T[-1] = S[7]$.

Элементы с 0 по 38 заполняются по рекурсивной формуле использующей как ключ, так и уже заполненные ранее значения T :

$$T[i] = (T[i-7] \text{ XOR } T[i-2] \text{ ROL } 3) \text{ XOR } k[i \bmod n] \text{ XOR } i \quad (i=0 \dots 38)$$

В последнюю 39-ю ячейку массива T помещается исходная длина ключа n с той целью, чтобы два разных по длине ключа не могли создать одинаковый массив T , и, как следствие, одинаковый материал ключа. Подобная ситуация может произойти, например, если первый ключ — 256-битный и состоит из двух одинаковых половинок, совпадающих со вторым 128-битным ключом.

По окончании заполнения массива T первые семь ячеек с отрицательными номерами отбрасываются. Их предназначение было только в исполнении рекурсивной формулы для малых значений счетчика i . В дальнейшем они использоваться не будут.

На втором этапе (перемешивании) весь массив T семикратно преобразуется по рекурсивной формуле, очень напоминающей некоторые хэш-функции:

Цикл по j от 0 до 7

Начало j

Цикл по i от 1 до 39

Начало i

$T[i] = (T[i] + S[LOW_9(T[i - 1])]) \text{ ROL } 9$

$T[0] = (T[0] + S[LOW_9(T[39])]) \text{ ROL } 9$

Конец i

Конец j

Функция LOW_9 означает взятие только младших 9 бит результата (табличная подстановка S имеет 9 бит на входе и 32 бита на выходе).

На третьем этапе производится перестановка, расставляющая далеко друг от друга машинные слова, находившиеся рядом в массиве T . В результате выполнения данного этапа создается массив материала ключа k из 40 32-битных машинных слов:

$$k[(7*i) \bmod 40] = T[i] \quad (i = 0...30)$$

Четвертый (один из самых сложных) этапов расширения ключа отвечает за то, чтобы на нечетных позициях в массиве k не оказались «слабые», с точки зрения операции умножения, числа. Преобразованию и проверке подвергаются все нечетные ячейки массива k с 5-й по 35-ю включительно. Первоначально два младших бита у каждого подобного элемента устанавливаются в «1» ($k[i] = k[i] \text{ OR } 3$), а их исходное значение (возможные значения 0, 1, 2, 3) запоминается перед этим в переменной $L[i]$ (оно потребуется на следующем этапе).

Это обеспечивает хотя бы минимально приемлемое перемешивание нескольких младших бит основного (неключевого) сомножителя в результате умножения.

Затем для каждого проверяемого элемента $k[i]$ создается битовая маска $M[i]$, в которой единицами отмечены те места в элементе $k[i]$, где идут подряд 10 или больше «нулей» или «единиц». Такие места в ключевом сомножителе $k[i]$ приведут к тому, что корреляция между некоторыми битами неключевого сомножителя и результатом умножения будет неприемлемо высока. В полученной маске $M[i]$ сбрасываются в «0» два младших бита, а также те биты,

которые стояли на границе (в начале и в конце) длинных последовательностей нулей или единиц, например:

Ключевой элемент $k[i]$ 01011111111111101100000000001101
Первоначальная маска $M[i]$ 00011111111111100011111111110000
Маска после сброса бит $M[i]$ 00001111111111100000111111110000

Наконец, производится собственно коррекция элемента $k[i]$ по следующей формуле:

$$k[i] = k[i] \text{ XOR } (S[265 + L[i]] \text{ ROL LOW_5 } (K[I + 3]))$$

Функция LOW_5 означает взятие младших 5 бит от числа. Элементы табличной подстановки с номерами 265...268 (напомним, что $L[i]$ может принимать только значения 0, 1, 2, 3) выбраны авторами как обладающие наиболее удачными для подобной коррекции вероятностными характеристиками.

MARS: анализ

Основные преимущества алгоритма по сравнению с другими претендентами заключаются в специфичной процедуре расширения ключа и в парадигме «множество операций». Используемое расширение ключа позволяет, во-первых, применять алгоритм с ключом произвольной длины от 32 до 1248 бит (с шагом в 32 бита). Во-вторых, мощное криптографическое преобразование еще на этапе расширения материала ключа (по стойкости оно не уступает некоторым хэш-функциями) позволяет использовать в качестве ключа даже сильно коррелированные между собой данные, например, текст в формате ASCII (пароль – до 156 символов) или Unicode (до 78 символов).

Использование в самом ядре криптографического преобразования множества разноплановых арифметических и логических операций действительно с большой вероятностью поможет выстоять алгоритму против новых еще не открытых схем криптоанализа. Если, конечно, не будет реализован какой-либо качественный прорыв в криптотехнологиях.

К недостаткам алгоритма исследователи относят достаточно большой размер табличных подстановок ($512 \times 4 = 2048$ байт), что может сказаться на реализации в системах с ограниченными ресурсами ПЗУ. Процедура расширения ключа и 32-разрядное умножение также достаточно сложны для слабых микроципов, по-

добных смарт-картам и контроллерам. Из-за применения умножения и сдвигов на переменное число бит шифр потенциально не стоек к атакам по потребляемой мощности и времени исполнения. Алгоритм слабо поддается распараллеливанию.

2.5.2. RC6

RC6: структура

Шифр RC6 представлен группой разработчиков во главе с мировой фигурой в области криптографии Рональдом Ривестом (Ronald Rivest).

Алгоритм разработан за довольно короткий для криптографии период на основе предыдущего популярного блочного шифра RC5. Изменения коснулись длины ключа, требуемой от конкурсантов AES, размера блока и некоторых незначительных моментов в основной части.

Алгоритм RC6 исключительно прост. Его четкая краткая структура и отсутствие табличных подстановок делают его очень привлекательным для реализации как криптографической прослойки в программных продуктах. Шифр представляет собой сеть Файстеля из 20 раундов с 4 ветвями смешанного типа – результат образующих функций, вычисляемых от четных ветвей, накладывается на нечетные ветви, затем происходит обмен ветвей местами. Общий вид алгоритма (с входным и выходным забеливанием) приведен на рис. 19, а структура одного раунда – на рис. 20.

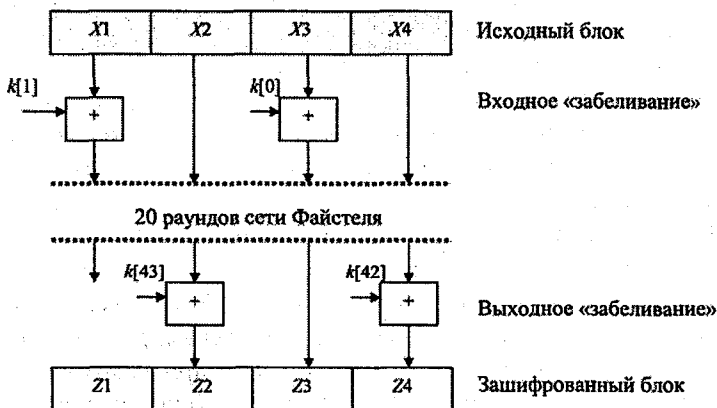


Рис. 19. Общий вид алгоритма RC6

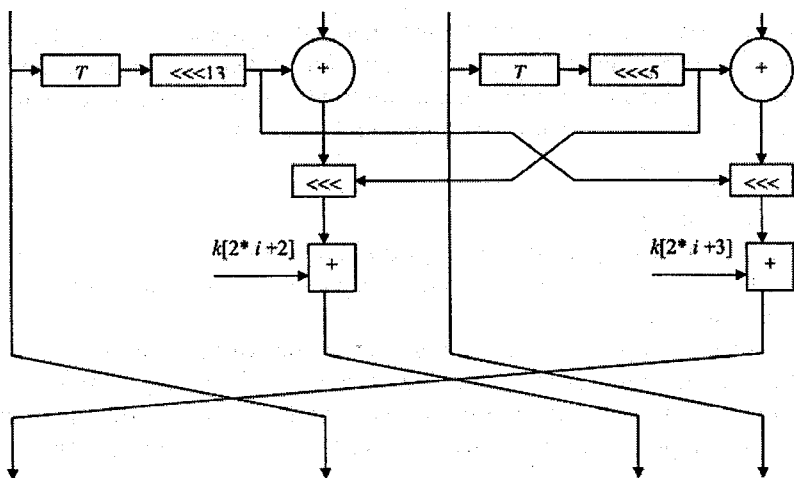


Рис. 20. Раунд сети фэй алгоритма RC6

Преобразование T представляет собой функцию

$$T(X) = \text{LOW_32} (X \times (2 \times X + 1))$$

с 32-разрядным входом X и 32-разрядным результатом (возвращаются только 32 младших бита от результата умножения).

Операция умножения обладает хорошими перемешивающими свойствами, и в RC6 это свойство используется на максимальном уровне. Как видно из схемы, в качестве параметра циклического сдвига на переменное число бит поступает результат умножения, циклически смещенный влево на 5 бит, т.е. реально величину переменного сдвига определяют 27...31-й биты результата 32-битного умножения – а именно эти биты находятся в центре 64-битного произведения и, следовательно, зависят от всех бит входного параметра X .

Необходимо отметить и такую особенность преобразования T как биективность, т.е. обратимость – взаимно однозначное соответствие между операндом и результатом. Это довольно неочевидный и редкий факт для операции умножения, да еще и с отсечением некоторой части результата. Однако при некоторых наборах констант (и в данном случае тоже) подобная операция биективна и следующее за ней наложение результата умножения опе-

рацией XOR на изменяемую ветвь «переносит» на нее все 32-бита информационной емкости неизменяемой ветви. Применение в качестве образующей функции сети Файстеля именно обратимого преобразования хотя и не обязательно, но считается хорошим тоном в криптографии.

Зашифрование алгоритма RC6 производится инвертированием порядка выполняемых в сети Файстеля действий и заменой операции сложения на вычитание. Из-за смешанного типа применяемой сети и простоты алгоритма в целом дешифрование RC6 проще реализовывать отдельным блоком программного блока.

RC6: расширение ключа

Алгоритм расширения ключа в RC6 сравним по простоте и скорости исполнения с основным преобразованием. Первоначально ключ (который теоретически может иметь произвольную длину) выравнивается по 32-битной границей нулями. Пусть полученный в результате массив K имеет длину в 32-битных машинных слова и нумеруется соответственно с $K[0]$ по $K[c-1]$. Массив расширенного материала ключа назовем традиционно k , для 20 раундов и забеливания потребуется 44 элемента: $k[0] \dots k[43]$. В процедуре расширения используются две константы $P = B7E15163_{16}$ и $Q = 9E3779B9_{16}$, представляющие собой старшие 32 бита двоичной записи дробной части чисел $e = 2,71828182845 \dots$ (основание натурального логарифма) и $\phi = 1,61803398875 \dots$ (золотое сечение) соответственно.

На первом этапе массив k заполняется константами по формуле:

$$k[i] = P + i \times Q \quad (i = 0 \dots 43).$$

На втором этапе производится рекурсивное перемешивание между собой массивов K и k с помощью двух вспомогательных 32-битных переменных A и B . Алгоритм перемешивания выглядит следующим образом:

$A = 0; B = 0; i = 0; j = 0$

Цикл по x от 1 до 44×3

Начало $_x$

$k[i] = (k[i] + A + B) \text{ ROL } 3; B = k[i];$

$i = (i + 1) \bmod 44$

$K[j] = (K[j] + A + B) \text{ ROL } (A + B); A = K[j]$

$j = (j + 1) \bmod c$

конец $_x$

Схематично данный алгоритм можно изобразить, как показано на рис.21, если представить, что массивы K и k замкнуты в виде колец (первое с периодом оборота c , второе – с периодом 44). Как видно, процедура перемешивания алгоритмически проста – в ней используются только арифметическое сложение и циклические сдвиги. Однако с точки зрения криптоанализа подобное трехкратное рекурсивное перемешивание материала расширения с ключом (а главное, при наличии сдвигов на переменное число бит) очень трудно анализируется.

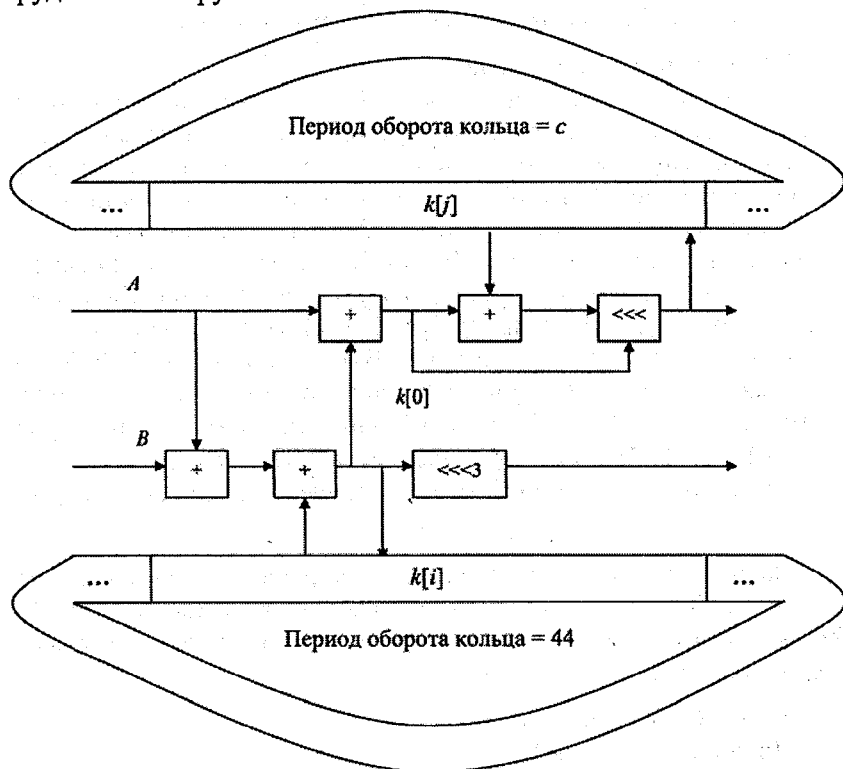


Рис. 21. Раунд расширения ключа RC6.

RC6: анализ

Основными преимуществами RC6 над другими претендентами исследователи единодушно называли очень высокую скорость работы шифра на 32-разрядных процессорах, притом, что крипто-

стойкость алгоритма практически «гарантируется» его предшественником – RC5 (прошедшим гораздо больший период безуспешных попыток взлома, чем большинство других претендентов). Сочетание этих двух факторов формирует для RC6 очень высокий коэффициент «стойкость/быстродействие», а он, как уже упоминалось, является основополагающим в криптографии. Среди других преимуществ называется сверхмалый объем кода как основной части алгоритма, так и процедуры расширения ключа.

2.5.3. *Rijndael*

Rijndael: структура

Данный блочный шифр развивает нетрадиционную для последнего десятилетия структуру: не сеть Файстеля, а прямоугольные KASLT-сети. Эта структура получила свое название из-за того, что шифруемый блок представляется в виде прямоугольника, например 4×4 или 4×6 байт, и затем над ним построчно, по столбцам и побайтно производятся криптопреобразования.

Rijndael – это шифр с вариативными размером блока и длиной ключа. Единственные налагаемые требования на эти два параметра – их кратность 32 битам. Сама структура и последовательность операций позволяют выполнять данный алгоритм эффективно как на 8-битных, так и на 32-битных процессорах. Размер блока далее определяется как $Nb \times 32$, длина ключа – $Nk \times 32$. Количество раундов сети, необходимое для стойкого шифрования, зависит от этих двух параметров.

Все преобразования в шифре имеют строгое математическое обоснование. В структуре алгоритма заложена возможность параллельного исполнения некоторых операций, что на многопроцессорных рабочих станциях может еще поднять скорость шифрования в 4 раза.

Алгоритм состоит из некоторого количества раундов (от 10 до 14 – это зависит от размера блока и длины ключа), в которых последовательно выполняются следующие операции:

- ByteSub – табличная подстановка 8×8 бит (рис. 22),

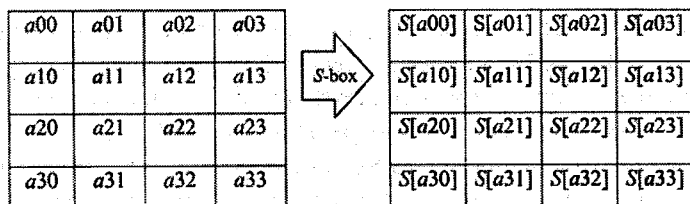


Рис. 22. Табличная подстановка в Rijndael

– ShiftRow – сдвиг строк в двумерном массиве на различные смещения (рис.23),

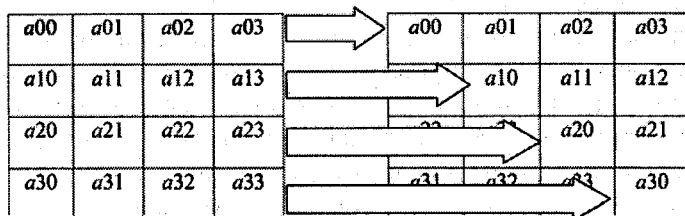


Рис. 23. Сдвиг строк в Rijndael

– MixColumn – математическое преобразование, перемешивающее данные внутри столбца (рис.24),

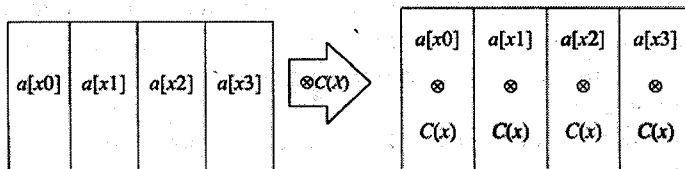


Рис. 24. Линейное перемешивание в Rijndael

– AddRoundKey – добавление материала ключа операций XOR (рис.25).

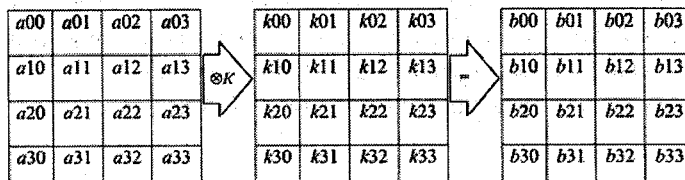


Рис. 25. Добавление материала ключа в Rijndael

В последнем раунде операция перемешивания столбцов отсутствует, что делает всю последовательность операций симметричной.

Дешифрование производится выполнением в обратном порядке криптопримитивов, инверсных к шифрующим. Наложение материала ключа обратимо самому себе. Сдвиг строк производится циклически в обратную сторону. Табличная подстановка S заменяется на инверсную ей S^{-1} . Полином линейного перемешивания $C(x) = 3x^3 + x^2 + x + 2$ выбран так, что к нему существует инверсный полином $D(x) = 0B_{16} \times x^3 + 0D_{16} \times x^2 + 09_{16} \times x + 0E_{16}$, $C(x) * D(x) = 1$.

Rijndael: анализ

К достоинствам алгоритма необходимо отнести очень хорошее быстроедействие на всех платформах от 8- до 64-битных. Выше по скорости исполнения только Serpent на аппаратной реализации и RC6 на 32-битной платформе – причина этого в очень сильной оптимизации данных двух алгоритмов под конкретные архитектуры ЭВМ.

Преимущества, относящиеся к аспектам реализации:

- Rijndael может выполняться быстрее, чем обычный блочный алгоритм шифрования. Выполнена оптимизация между размером таблицы и скоростью выполнения;
- Rijndael можно реализовать в смарт-карте в виде кода, используя небольшой RAM и имея небольшое число циклов;
- преобразование раунда допускает параллельное выполнение, что является важным преимуществом для будущих процессоров и специализированной аппаратуры;
- алгоритм шифрования не использует арифметические операции, поэтому тип архитектуры процессора не имеет значения;
- переменная длина блока: длины блоков от 192 до 256 бит позволяют создавать хэш-функции без коллизий, использующие Rijndael в качестве функции сжатия. Длина блока 128 бит сегодня считается для этой цели недостаточной;
- разработка позволяет специфицировать варианты длины блока и длины ключа в диапазоне от 128 до 256 бит с шагом в 32 бита;

– хотя число раундов Rijndael зафиксировано в данной спецификации, в случае возникновения проблем с безопасностью он может модифицироваться и иметь число раундов в качестве параметра.

К недостаткам шифра исследователи относят необходимость разработки двух независимых процедур – шифрования и дешифрования (объединить их, как, например, в сети Файстеля, уже невозможно).

2.5.4. TwoFish

Алгоритм разработан компанией Counterpain Security Systems, возглавляемой Брюсом Шнайером (англ. Bruce Schneier). Предыдущая программная разработка этой фирмы, называвшаяся BlowFish, являлась и до сих пор является признанным криптостойким алгоритмом.

Twofish разрабатывался специально с учетом требований и рекомендаций для AES:

- 128-битный блочный симметричный шифр;
- отсутствие слабых ключей;
- эффективная программная (в первую очередь на 32-битных процессорах) и аппаратная реализация;
- гибкость (возможность использования дополнительных длин ключа, использование в поточном шифровании, хэш-функциях и т. д.);
- простота алгоритма.

Алгоритм Twofish возник в результате попытки модифицировать алгоритм Blowfish для 128-битового входного блока. Новый алгоритм должен был быть легко реализуемым аппаратно (в том числе использовать таблицы меньшего размера), иметь более совершенную систему расширения ключа и иметь однозначную функцию F .

В алгоритме TwoFish разработчики оставили некоторые удачные решения из проекта-предшественника, кроме этого произвели тщательные исследования по перемешиванию данных в сети Файстеля.

Единственным нарицанием, поступившим в адрес TwoFish от независимых исследователей, является тот факт, что при расшире-

нии материала ключа в алгоритме используется сам же алгоритм. Двойное применение блочного шифра довольно сильно усложняет его анализ. Именно сложность структуры алгоритма и анализа, а также достаточно медленное время выполнения по сравнению с Rijndael на большинстве платформ, сыграло не в его пользу.

Алгоритм Twofish не запатентован и может быть использован кем угодно без какой-либо платы или отчислений. Он используется во многих программах шифрования, хотя и получил меньшее распространение, чем Blowfish.

TwoFish: структура

Алгоритм представляет собой сеть Файстеля смешанного типа: первая и вторая ветвь на нечетных раундах производят модификацию третьей и четвертой, на четных раундах ситуация меняется на противоположную. В алгоритме используется криптопреобразование Адамара (Pseudo-Hadamard Transform, PHT) – обратимое арифметическое сложение первого потока со вторым, а затем второго с первым.

Twofish разбивает входной 128-битный блок данных на четыре 32-битных подблока, над которыми, после процедуры входного забеливания, производится 16 раундов преобразований. После последнего раунда выполняется выходное забеливание.

Общая структура шифра и схема образующей функции приведена на рис. 26.

Основными перемешивающими элементами являются табличные подстановки 8×8 бит и умножение на матрицу. Оба этих компонента достаточно сложны по своей структуре. Во-первых, в алгоритме используется табличная подстановка, зависящая от ключа: над данными производится сложное преобразование, зависящее от ключа, которое для ускорения вычислений можно представить в виде табличной подстановки. Второй сложной перемешивающей операцией в Twofish является умножение на матрицу MDS (maximum distance separable), выбор коэффициентов которой был сделан на основе некоторых идей из теории помехоустойчивого кодирования. Сама матрица представлена на рис. 27.

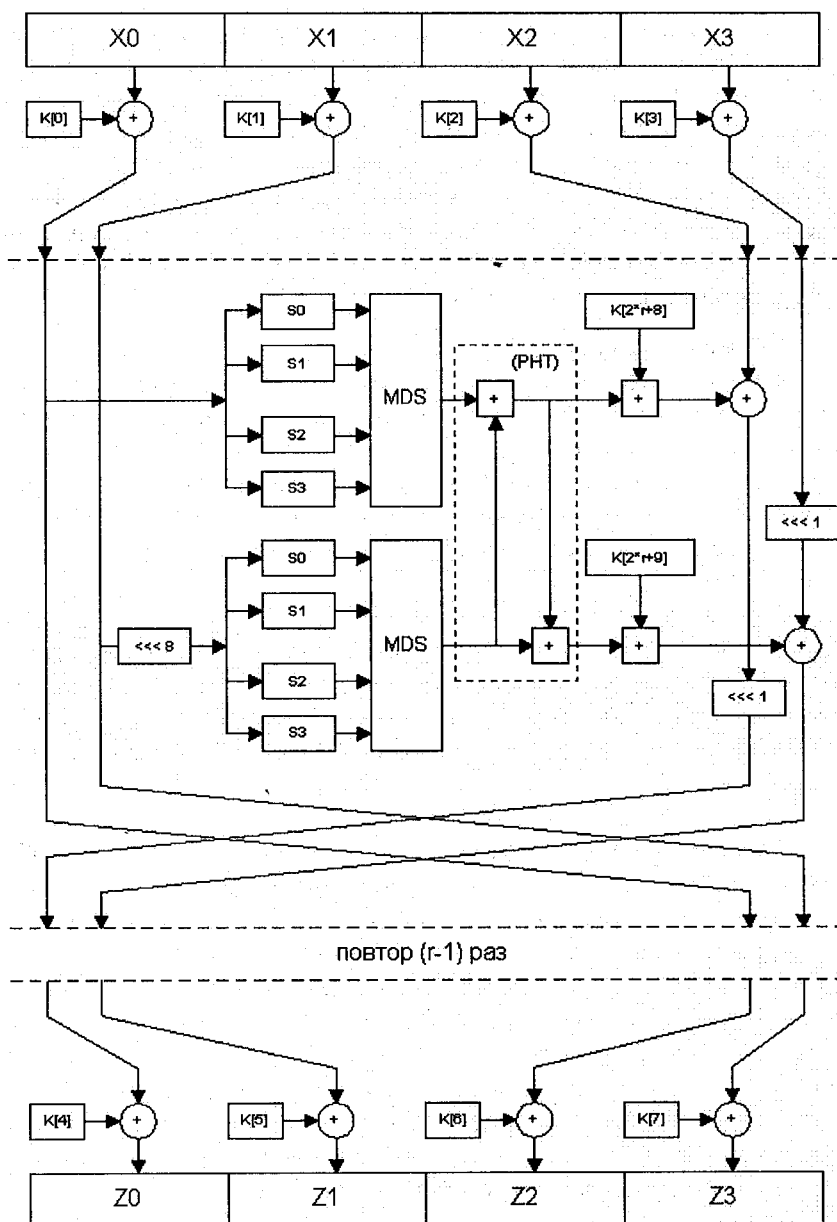


Рис. 26. Структура TwoFish

01	EF	5B	5B
5B	EF	EF	01
EF	5B	01	EF
EF	01	EF	5B

Рис. 27. Матрица MDS алгоритма TwoFish

Умножение производится в полиномиальной форме по модулю 2. В качестве образующего для операции полиномиального умножения выбран следующий полином: $V(x) = x^8 + x^6 + x^3 + 1$.

TwoFish: расширение ключа

Алгоритм имеет самую сложную из всех претендентов процедуру расширения ключа. Кроме того, на этапе расширения ключа создаются 4 табличные подстановки S_0, S_1, S_2, S_3 каждая разрядностью 8×8 . На начальном этапе ключ произвольной длины дополняется нулями до ближайшей большей длины: 128, 192 или 256 бит. Величина j определяется как $N/64$ (N – длина ключа в битах), т.е. j может принимать значения 2, 3 или 4.

На следующем этапе формируются три вектора длины j из 32-разрядных машинных слов из ключа:

$$M_0 = (K[0], K[2], \dots)$$

$$M_1 = (K[1], K[3], \dots)$$

Для получения массива S ключ рассматривается уже как массив из 64-разрядных значений. Каждое из j значений умножается в полиномиальной форме по модулю 2 на трансформирующую матрицу RS разрядностью 4×8 байт, созданную на основе кода Рида-Соломона (рис. 28).

01	A4	55	87	5A	58	DB	9E
A4	56	82	F3	1E	C6	68	E5
02	A1	FC	C1	47	AE	3D	19
A4	55	87	5A	58	DB	9E	04

Рис. 28. Матрица RS алгоритма TwoFish

Полученный после каждого умножения 4-байтный вектор расценивается как 32-битное значение и помещается в массив S . Значения размещаются в массиве S в обратном порядке.

Таким образом, векторы M_0 , M_1 , S составляют базис процедуры расширения ключа.

TwoFish: анализ

Изучение Twofish с сокращенным числом раундов показало, что алгоритм обладает большим запасом прочности и, по сравнению с остальными финалистами конкурса AES, он оказался самым стойким. Среди недостатков называются слабая распараллеливаемость алгоритма, потенциальная уязвимость к атаке по потребляемой мощности, высокая криптосложность и низкая скорость процедуры расширения ключа.

2.5.5. Serpent

Алгоритм разработан группой ученых из нескольких исследовательских центров мира.

Шифр является продолжателем семейства SP-сетей. В целом Serpent представляет собой очень консервативный по своей структуре шифр, чего и добивались авторы, так как за десятки лет криптоатак на DES так и не было найдено действительно эффективного метода его взлома.

Serpent: структура

Алгоритм представляет собой сеть Файстеля для четырех ветвей смешанного типа: 2 четные ветви изменяют совместно значения нечетных, затем меняются местами. В качестве криптопреобразований используются только «исключающее ИЛИ», «табличные подстановки» и «битовые сдвиги». Алгоритм состоит из 32 раундов. Сами раунды составлены таким образом, что добавление к ветвям материала ключа на первом и последнем раундах образует входное и выходное забеливание.

Предварительно из ключа произвольной длины от 1 до 256 бит создается 33 блока материала ключа $k[0]...k[32]$, каждая размеров в 128 бит.

Циклы с 0 по 30 состоят из:

- наложения на блок материала ключа $k[i]$ операцией XOR;
- 32-х табличных подстановок 4×4 бита;
- линейного преобразования, задействующего все биты блока.

В алгоритме используются 8 табличных подстановок $S[0]...S[7]$, каждая 4×4 бита.

В нулевом раунде используется 32 раза подстановка $S[0]$, в первом – $S[1]$, ..., в седьмом – $S[7]$, в восьмом – снова $S[0]$, и так далее по циклу. Линейное преобразование задано перечислением бит входного блока до преобразования, от которых зависит каждый конкретный бит блока после преобразования. Например, нулевой бит результата преобразования получается применением операции XOR над 16, 52, 56, 70, 83, 94 и 105-м битами исходного блока, первый бит – применением операции XOR над 72, 114 и 125-м битами и т.д.

Последний 31-й раунд отличается тем, что вместо линейного преобразования на результат 31-й табличной подстановки накладывается операцией XOR 32-й блок материала ключа $k[32]$. Всед за последним раундом производится перестановка бит, обратная той, которая и выполнялась в самом начале шифрования.

Общая схема алгоритма Serpent приведена на рис. 29.

Serpent: расширение ключа

Расширение ключа в алгоритме производится в три этапа. Первоначально к ключу произвольной длины добавляется справа бит «1», а затем столько «0», сколько требуется до получения 256-битного массива.

На втором этапе получившиеся блоки записываются в массив 32-битных чисел $W[i]$ под индексами от (-8) до (-1) . Последующие значения массива W с 0-го по 131-й вычисляются по формуле:

$$W[i] = (W[i-8] \text{ XOR } W[i-5] \text{ XOR } W[i-3] \text{ XOR } W[i-1] \text{ XOR } F \text{ XOR } i) \text{ ROL } 11,$$

где $F=9E3779B9_{16}$ – двоичная запись старших бит дробной части золотого сечения – ϕ .

На третьем этапе над 132 машинными словами массива W производится табличная подстановка. Подряд стоящие блоки по 4 машинных слова и составляют одно 128-битное значение для материала ключа.

Serpent: анализ

Алгоритм имеет самую высокую среди всех алгоритмов, представленных в финале, скорость в полностью аппаратной реализации (на простейших электронных элементах). В программной

реализации алгоритм использует только логические операции и сдвиги на фиксированное количество бит – это делает его стойким к атакам по потребляемой мощности и времени.

Среди недостатков алгоритма следует назвать самую низкую среди алгоритмов скорость исполнения на 32-разрядных платформах: парадигма побитовых операций проигрывает более современным криптопримитивам в скорости исполнения. Вторым недостатком является слабая возможность рапараллеливания алгоритма.

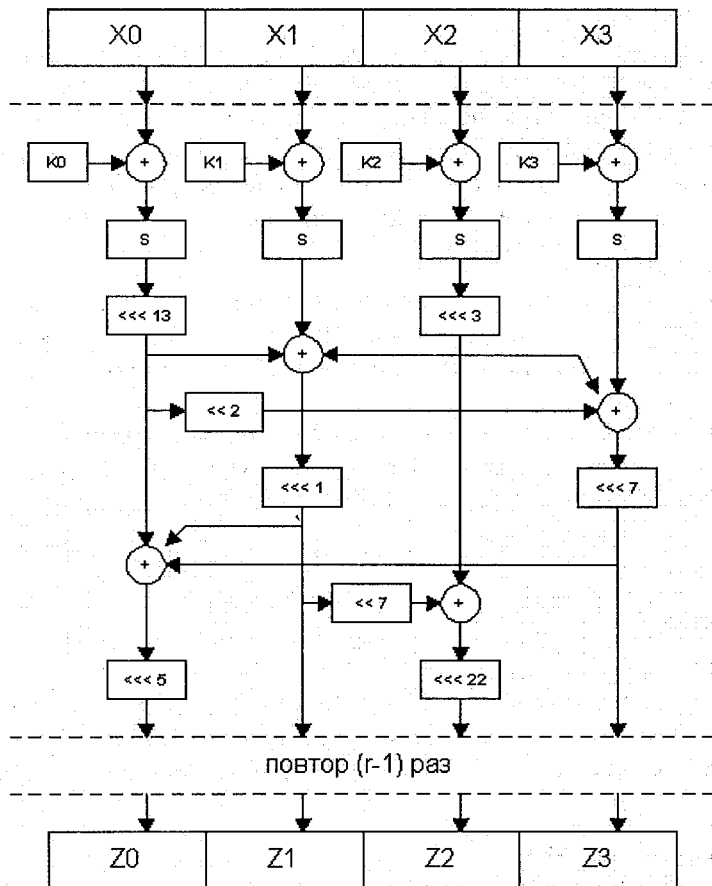


Рис. 29. Алгоритм Serpent

2.5.6. Итоги конкурса AES

Сравнение качественных характеристик алгоритмов конкурса AES приведено в табл. 2.

Таблица 2

Преимущества	Алгоритмы				
	Rijndael	Serpent	TwoFish	RC-6	MARS
Бысродействие: аппаратная реализация	+	+			
Программная реализация: на слабых ВС на мощных ВС	+	+		+	
Этап расширения ключа	+		+		
Распараллеливаемость	+				
Этап дешифрования			+	+	+
Запас крипто- стойкости	Оптимум	Завышен	Завышен	Оптимум	Завышен
Количество голо- сов	86	59	31	23	13

2 октября 2000 года алгоритм RIJNDAEL был официально оглашен победителем конкурса и получил второе наименование AES – Advanced Encryption Standard.

3. СИСТЕМЫ ШИФРОВАНИЯ С ОТКРЫТЫМ КЛЮЧОМ

3.1. Основные принципы

Симметричная криптография, обладающая очень простой и понятной схемой ключей: отправитель и получатель знают один общий секрет, и третьим лицам он не доступен, скрывает в себе подводные камни при значительном увеличении числа абонентов, желающих шифровать взаимную переписку. Если пользователь хочет конфиденциально вести дела с N партнерами, ему необходимо иметь N секретных ключей, а общее количество ключей в системе растет от числа абонентов по формуле $N \times (N - 1) / 2$. Кроме того, появляются вопросы доверенной доставки ключей абонентам и апшелируемости, т.е. ситуации, когда получатель, дешифровав сообщение, может внести в него изменения, а затем ссылаться на документ, якобы уже полученный в таком виде.

Ассиметричная криптография решает этот вопрос кардинально иначе: во-первых, количество ключей сокращается до $2 \times N$. Кроме данного факта асимметричная криптография обладает еще целым рядом свойств, которые сделали ее идеальной технологией для шифрования общения двух удаленных друг от друга абонентов.

В асимметричной криптографии применяется шифрование, при котором некая величина разделяется на два ключа. Один из них делается открытым, а другой сохраняется в секрете. Сообщение шифруется с помощью открытого ключа получателя. Получатель может затем расшифровать его с помощью своего секретного ключа. Он может сделать то же самое для любого другого абонента, например, шифруя сообщение с помощью вашего открытого ключа, чтобы вы могли расшифровать его с помощью своего секретного ключа (рис.30). Суть в том, что не требуется знать чей-то секретный ключ, чтобы послать защищенное сообщение. Для этого применяется открытый ключ, который не нужно держать в секрете (на самом деле, его можно публиковать наравне с номером телефона). Используя открытый ключ получателя, вы знаете, что

только этот человек может расшифровать сообщение при помощи своего секретного ключа. Такой подход позволяет двум корреспондентам безопасно общаться без предварительного обмена ключами.

Таким образом, в качестве ключа зашифровывания должен использоваться открытый ключ получателя, а в качестве ключа расшифровывания – его секретный ключ.

Секретный и открытый ключи генерируются попарно. Секретный ключ должен оставаться у его владельца и быть надежно защищен от несанкционированного доступа (аналогично ключу шифрования в симметричных алгоритмах). Копия открытого ключа должна находиться у каждого абонента криптографической сети, с которым обменивается информацией владелец секретного ключа.

На рис. 30 приведены основные принципы асимметричной криптографии.

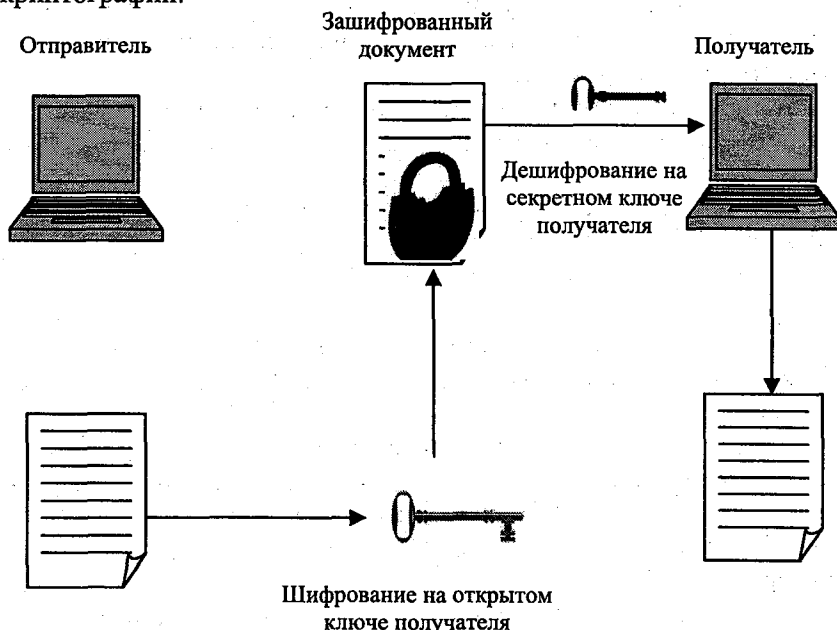


Рис. 30. Распределение ключей в асимметричной криптографии

Идея асимметричной криптографии была предложена американскими учеными У. Диффи и М. Хеллманом в 1976 году. Они сформулировали требования, выполнение которых обеспечивает безопасность асимметричной криптосистемы:

- Вычисление пары ключей (K_B, k_B) получателем B должно быть простым.

- Отправитель A , зная открытый ключ K_B и сообщение M , может легко вычислить криптограмму $C = E_{K_B}(M)$.

- Получатель B , используя секретный ключ k_B и криптограмму C , может легко восстановить исходное сообщение $M = D_{k_B}(C)$.

- Противник, зная пару (K_B, C) , при попытке вычислить исходное сообщение M наталкивается на непреодолимую вычислительную проблему.

- Противник, зная открытый ключ K_B , при попытке вычислить секретный ключ k_B также наталкивается на неразрешимую вычислительную проблему.

Концепция асимметричных криптографических систем с открытым ключом основана на применении однонаправленных функций.

Однонаправленной функцией называется функция $F(X)$, обладающая двумя свойствами:

- существует алгоритм вычисления значений функции $Y = F(X)$;

- не существует эффективного алгоритма обращения (инвертирования) функции F (т.е. не существует решения уравнения $F(X) = Y$ относительно X).

В качестве примера однонаправленной функции можно указать целочисленное умножение. Прямая задача – вычисление произведения двух очень больших целых чисел P и Q , т. е. нахождение значения $N = P \times Q$ – относительно несложная задача для компьютера.

Обратная задача – факторизация, или разложение на множители большого целого числа, т.е. нахождение делителей P и Q большого целого числа $N = P \times Q$, является практически неразрешимой при достаточно больших значениях N .

Процесс передачи зашифрованной информации в асимметричной криптосистеме осуществляется следующим образом.

Подготовительный этап:

- абонент B генерирует пару ключей: секретный ключ k_B и открытый ключ K_B ;

- открытый ключ K_B посылается абоненту A и остальным абонентам (или делается доступным, например, на разделяемом ресурсе).

Использование – обмен информацией между абонентами A и B :

- абонент A зашифровывает сообщение с помощью открытого ключа K_B абонента B и отправляет шифртекст абоненту B ;

- абонент B расшифровывает сообщение с помощью своего секретного ключа k_B . Никто другой (в том числе абонент A) не может расшифровать данное сообщение, так как не имеет секретного ключа абонента B . Защита информации в асимметричной криптосистеме основана на секретности ключа k_B получателя сообщения.

Как и в случае симметричных криптографических систем, с помощью асимметричных криптосистем обеспечивается не только конфиденциальность, но также подлинность и целостность передаваемой информации. Подлинность и целостность любого сообщения обеспечивается формированием электронной цифровой подписи (ЭЦП) этого сообщения и отправкой в зашифрованном виде сообщения вместе с цифровой подписью. Проверка соответствия подписи полученному сообщению после его предварительного расшифровывания представляет собой проверку целостности и подлинности принятого сообщения.

Преимущества асимметричных криптографических систем перед симметричными криптосистемами:

- в асимметричных криптосистемах решена сложная проблема распределения ключей между пользователями, так как каждый пользователь может сгенерировать свою пару ключей сам, а открытые ключи пользователей могут свободно публиковаться и распространяться по сетевым коммуникациям;

- исчезает квадратичная зависимость числа ключей от числа пользователей; в асимметричной криптосистеме число используемых ключей связано с числом абонентов линейной зависимостью

(в системе из N пользователей используются $2N$ ключей), а не квадратичной, как в симметричных системах;

- асимметричные криптосистемы позволяют реализовать протоколы взаимодействия сторон, которые не доверяют друг другу, поскольку при использовании асимметричных криптосистем закрытый ключ должен быть известен только его владельцу.

- Недостатки асимметричных криптосистем:

- на настоящий момент нет математического доказательства необратимости используемых в асимметричных алгоритмах функций;

- асимметричное шифрование существенно медленнее симметричного, поскольку при шифровании и расшифровке используются весьма ресурсоемкие операции. По этой же причине реализовать аппаратный шифратор с асимметричным алгоритмом существенно сложнее, чем реализовать аппаратно симметричный алгоритм;

- необходимость защиты открытых ключей от подмены.

Асимметричная криптография может применяться для:

- направленного шифрования;

- формирования и проверки ЭЦП;

- выработки общего секрета.

Вскоре после публикации У. Диффи и М. Хеллманом другая группа из трех человек разработала практическое приложение данной идеи. Эта система для шифрования с открытым ключом была названа RSA по именам авторов: Ronald Rivest (Рональд Ривест), Adi Shamir (Ади Шамир) и Leonard Adleman (Леонард Адлеман).

3.2. Практические схемы асимметричного шифрования

3.2.1. Криптоалгоритм RSA

Структура RSA

Первый этап любого симметричного алгоритма – создание пары ключей – стоит для схемы RSA из следующих операций.

1. Выбираются два больших простых числа p и q .
2. Вычисляется n , равное $(p \times q)$.

3. Выбирается произвольное число e ($e < n$), такое, что наибольший общий делитель $\text{НОД}(e, (p-1) \times (q-1)) = 1$, т.е. e должно быть взаимно простым с числом $(p-1) \times (q-1)$.

4. Методом Евклида решается в целых числах уравнение $e \times d + (p-1) \times (q-1) \times y = 1$. Здесь неизвестными являются переменные e и y — метод Евклида как раз и находит множество пар (d, y) , каждая из которых является решением уравнения в целых числах.

5. Пара чисел (e, n) — публикуется как открытый ключ. Число d хранится в секрете — это и есть закрытый ключ, который позволяет читать все послания, зашифрованные с помощью пары чисел (e, n) .

Второй этап — собственно шифрование с помощью открытого ключа.

1. Отправитель разбивает свое сообщение на блоки, равные $k = [\log_2(n)]$ бит, где квадратные скобки обозначают взятие целой части от дробного числа. Подобный блок может быть интерпретирован как число из диапазона $(0; 2^k - 1)$.

2. Для каждого типа числа (назовем его m_i) вычисляется выражение $c_i = ((m_i)^e) \bmod n$. Блоки c_i и есть зашифрованное сообщение. Их можно без опасения передать по открытому каналу, поскольку операция возведения в степень по модулю простого числа является той самой трудноразрешимой математической задачей.

3. Третий этап — дешифрование послания с помощью секретного ключа. Частный случай теоремы Эйлера утверждает, что если число n представимо в виде произведения двух простых чисел p и q , то для любого x имеет место равенство:

$$(x^{(p-1) \times (q-1)}) \bmod n = 1.$$

Для дешифрования RSA-сообщения воспользуемся этой формулой. Возведем обе части ее части в степень $(-y)$: $(x^{(-y) \times (p-1) \times (q-1)}) \bmod n = 1^{(-y)} = 1$. После умножения обеих частей равенства на x получим

$$(x^{(-y) \times (p-1) \times (q-1) + 1}) \bmod n = 1 \times x = x.$$

Величина d была подобрана с помощью алгоритма Евклида так, что $e \times d + (p-1) \times (q-1) \times y = 1$, то есть $e \times d = 1 + (-y) \times (p-1) \times (q-1)$. Следовательно, в последнем выражении предыдущего аб-

заца мы можем заменить показатель степени на число $(e \times d)$. Получаем

$$(x^{(exd)}) \bmod n = (x^{(-y)(p-1) \times (q-1)+1}) \bmod n = 1 \times x = x.$$

То есть для того чтобы прочесть сообщение $c_i = ((m_i)^e) \bmod n$, достаточно возвести его в степень d по модулю n :

$$((c_i)^d) \bmod n = ((m_i)^{(exd)}) \bmod n = m_i.$$

Общий вид RSA системы приведен на рис. 31.

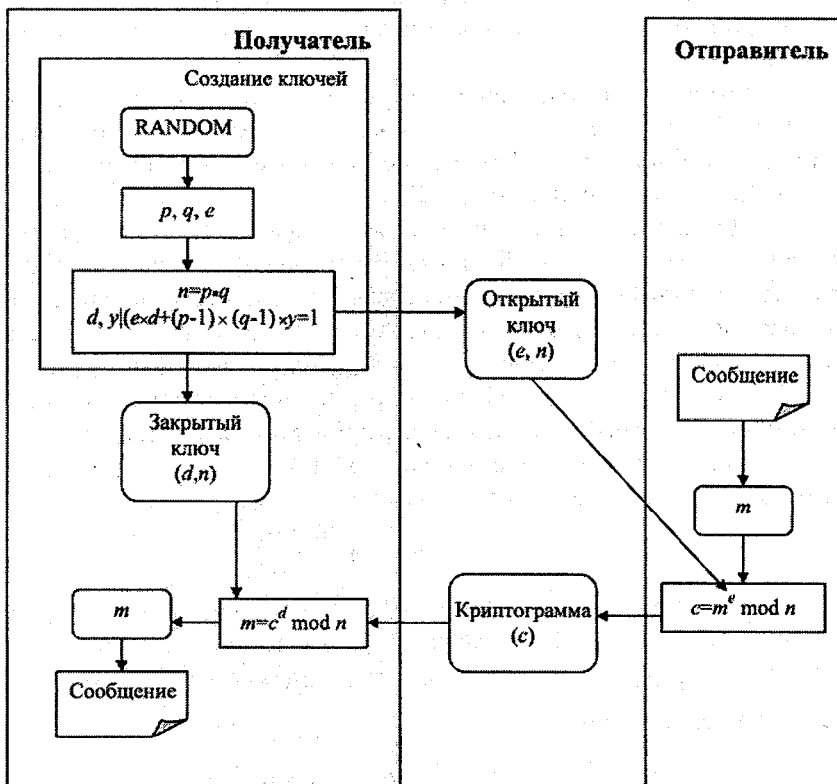


Рис. 31. Криптосистема RSA

Схема RSA: анализ

За время существования криптосистемы RSA были обнаружены алгоритмы взлома схемы при частных случаях его параметров. Поэтому в настоящее время принцип выбора простых чисел p и q — основы криптосхемы — пополнился рядом дополнительных ограничений;

- числа $(p+1)$ и $(q+1)$ должны содержать в своем разложении на множители большие простые делители;
- числа p и q должны очень близко совпадать по порядку длины;
- секретный ключ не должен в результате процедуры создания ключей оказаться очень маленьким. Если подобное условие случайно выполнилось, придется «перевыбирать» случайное число e и повторить процедуру вычисления d ;
- Каждый абонент сети должен использовать уникальное значение n .

В современной криптографии порогом раскрываемости RSA считается длина n в 512 бит. Эта величина находится на «границе доверия». Кроме нее используются криптосхемы с ключами в 768, 1024, 2048 бит.

Пример схемы RSA

Рассмотрим работу схемы RSA на небольших числах. Естественно, из-за малых значений p и q дополнительные требования на их выбор не выдержаны. Пусть $p = 5$, а $q = 11$, тогда значение $n = 55$. В качестве открытого ключа e выбираем число 7, таким образом, весь открытый ключ имеет вид ($e = 7, n = 55$). Вычислим закрытый ключ d : уравнение $e \times d + (p-1) \times (q-1) \times y = 1$ приобретает вид $7 \times d + 40 \times y = 1$ и имеет в целых числовых решение $d = 23, y = -4 = 51$. Таким образом, закрытым ключом является число 23.

Пусть произвольный отправитель хочет передать абоненту комбинацию бит 100111₂, ее числовой эквивалент: 39₁₀. Возведем 39 в степень открытого ключа $e = 7$ по модулю $n = 55$: $(39^7 \bmod 55) = 19$. Число 19 является шифрограммой и передается по открытому каналу связи. Получатель по приходу сообщения возводит его в степень $d = 23$: $(19^{23} \bmod 55) = 39$. Исходное послание восстановлено.

3.2.2. Криптоалгоритм Рабина

Схема Рабина похожа на RSA, но возводит сообщение m не в высокую степень e , а в квадрат: $c = m^2 \bmod n$ (число n так же как и в схеме RSA является произведением двух больших простых чисел p и q). На приемной стороне возможно декодирование сообщения и по схеме шифрования RSA, но чаще применяется несколько более быстрое извлечение квадратного корня по модулю n .

Процедура извлечения квадратного корня при известном разложении n на сомножители требует полиномиальное время, а при отсутствии подобной информации – гораздо более трудоемкая. Если p и q выбраны так, что остаток от деления на 4 и у первого, и у второго числа равен трем, то вычисление квадратного корня производится по следующим формулам:

- алгоритм Евклида решается в целых числах уравнения $a \times p + b \times q = 1$;
- $r = c^{(p+1)/4} \bmod p$;
- $s = c^{(q+1)/4} \bmod q$;
- $x = (a \times p \times s + b \times q \times r) \bmod n$;
- $y = (a \times p \times s - b \times q \times r) \bmod n$;
- корнями числа c являются 4 значения: $(\pm x \bmod n)$ и $(\pm y \bmod n)$.

Преимущество схемы Рабина заключается в уменьшении открытого ключа: он состоит из одного числа n , что значительно ускоряет процесс шифрования. Недостатком является получение в результате извлечения квадратного корня на приемной стороне четырех различных значений: m_1, m_2, m_3, m_4 . Одно из этих значений действительно является зашифрованным текстом, три других – издержки модульной арифметики. Это свойство требует добавлять перед шифрованием в сообщение какие-либо метки, т.е. применять избыточное кодирование с тем чтобы алгоритм на приемной стороне с высокой вероятностью мог распознать среди четырех значений действительно верный вариант. Схема Рабина представлена на рис. 32.

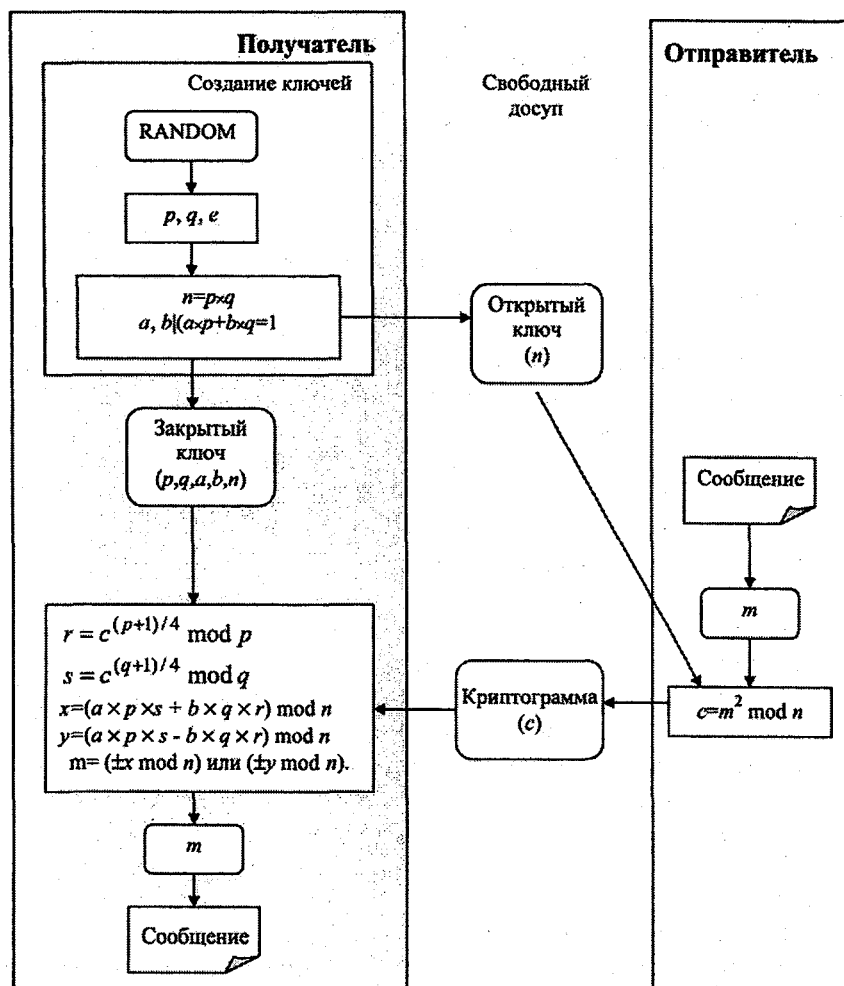


Рис. 32. Криптосхема Рабина

3.2.3. Криптоалгоритм Эль Гамаль

Эль Гамаль: структура

Асимметричная схема предложена автором как логичное продолжение алгоритма обмена ключами Диффи-Хелмана. Она использует ту же операцию возведения в степень по модулю простого числа, но трудноразрешимой для злоумышленника задачей ста-

вит отыскание не числа, которое возведено в степень, а то, в какую степень возведено известное число. Эта задача носит название проблемы дискретного логарифма.

Первый этап – создание пары ключей:

1. Выбирается произвольное, достаточно большое простое число p .
2. Для этого простого числа определяется любой образующий элемент, т.е. такое число a , при многократном возведении которого в степень по модулю p ($a^1 \bmod p, a^2 \bmod p, \dots$) будут перебираться все числа от 1 до $(p-1)$ включительно в произвольном порядке, но обязательно только по одному разу.
3. Генерируется произвольное случайное число x ($0 < x < p$) – это и есть закрытый ключ.
4. Вычисляется значение $b = a^x \bmod p$ – комбинация (a, p, b) представляет собой открытый ключ получателя.

Второй этап – шифрование с помощью открытого ключа:

1. Отправитель генерирует произвольное случайное число y ($0 < y < p$).
2. Помещает в начало шифрограммы число $(a^y \bmod p)$.
3. Вычисляет величину $k = (b^y \bmod p) = ((a^x \bmod p)^y \bmod p)$.
4. Используя некоторую, заранее оговоренную в данной реализации, часть k в качестве симметричного ключа для любого блочного шифра шифрует отправляемое сообщение.
5. Забывает числа y и k – надежно стерев их из оперативной памяти и других мест, куда они могут случайно попасть.

Третий этап – дешифрование послания с помощью секретного ключа:

1. По приходу зашифрованного сообщения получатель отделяет от пакета величину $(a^y \bmod p)$ и вычисляет на ее основе $((a^y \bmod p)^x \bmod p)$ – математика доказывает, что полученное число будет равно тому самому k , которое вычислил отправитель, так как в данной формуле операнды x и y можно менять местами.
2. Выделив из k ту же самую часть, что и отправитель, получатель дешифрует весь идущий далее пакет симметричным алгоритмом.

Общий вид системы приведен на рис. 33.

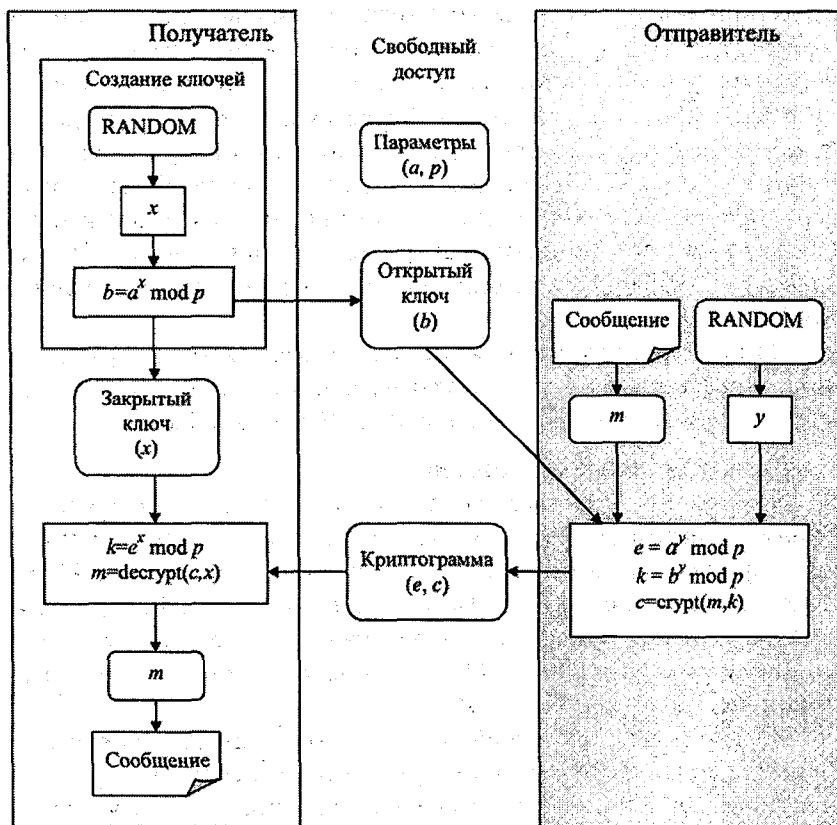


Рис. 33. Криптосистема Эль Гамаль

Схема Эль Гамаль: анализ

Проблема дискретного логарифмирования состоит в том, что, зная основание степени и получившейся после возведения результат по модулю простого числа, невозможно за полиномиальное время определить, в какую именно степень было возведено основание. В схеме Эль Гамаль потенциальный злоумышленник может получить значения a , p , $(a^x \bmod p)$ и $(a^y \bmod p)$. Однако из-за сложности определения чисел x и y «в чистом виде» у него не оказывается возможности вычислить значение $k = (a^{xy} \bmod p)$, которое ему необходимо для прочтения шифровки.

Относительно числа p криптоанализ выдвигает следующие требования. Число $(p - 1)$ должно содержать в разложении на множители большой простой делитель. В некоторых протоколах с участием схемы Эль Гамаль p выбирается как $(q \times 2 + 1)$, где q – простое число. При выборе x и y получателем и отправителем соответственно, естественно, должно выполняться требование к их информационной емкости. Для генерации этих чисел должен использоваться криптостойкий генератор случайных чисел. В противном случае злоумышленник может определить x или y полным перебором.

По криптостойкости в схеме Эль Гамаль 512-битное число p приравнивается к 56-битному симметричному ключу. Поэтому на практике применяются p длиной в 768, 1024 и 1536 бит.

Схема Эль Гамаль: пример

В качестве простого числа, порождающего циклическую группу, выберем $p = 11$, за образующий элемент примем число $a = 7$ (при возведении 7 в степень 1, 2, 3 и т.д. по модулю 11 последовательно проходят все 10 значений [7, 5, 2, 3, 10, 4, 6, 9, 8, 1]). Секретным ключом x выберем 6, параметр b принимает значение $b = (a^x \bmod p) = 7^6 \bmod 11 = 4$. В целом открытый ключ принимает вид $(a = 7, p = 11, b = 4)$.

Предположим, что некий абонент хочет передать сообщение. Он выбрал случайное число, не превосходящее p , например, $y = 9$. В начало шифрограммы помещается число $(a^y \bmod p) = 7^9 \bmod 11 = 8$. Кроме того, на основании y и открытого ключа отправитель вычисляет $k = b^y \bmod p = 4^9 \bmod 11 = 3$. Выбрав значение 3 или какие-либо его биты в качестве симметричного ключа, отправитель шифрует передаваемые данные и стирает величины 9 и 3 со своих накопителей.

Получатель по приходу пакета для вычисления $k = (a^y \bmod p)^x \bmod p$ возводит число 8 из заголовка шифрограммы в степень секретного ключа и получает $k = 8^6 \bmod 11 = 3$ – то же самое значение, которое использовал отправитель, шифруя собственно данные.

3.2.4. Криптоалгоритм Мак-Элиса

Схема Мак-Элиса, предложенная в 1978 г., основана на проблеме декодирования кода с исправлением ошибок. В общем слу-

чае линейный код не имеет полиномиального алгоритма декодирования, в то время как некоторые частные коды имеют эффективные схемы декодирования. В качестве секретного ключа выбирается матрица G разрядности $(k \times n)$, поражающая код с исправлением t ошибок, и две маскирующие матрицы (без знания их для злоумышленника код становится общим случаем линейного кода). Первая маскирующая матрица S разрядности $(k \times k)$ должна обладать свойством невырожденности. Вторая матрица P разрядности $(n \times n)$ является перестановочной, т.е. содержит закон перестановки из n бит. В качестве открытого ключа вычисляется матрица $W = S \times G \times P$, все абоненты сети знают в отношении получателя только матрицу W и то, какой степени ошибку может исправить эта матрица.

При отправке сообщения абонент представляет сообщение M в виде двоичного числа разрядности k бит и выбирает случайный вектор ошибки Z , содержащий в двоичной записи числа не более t единиц. Шифрограмма получается по формуле $C = M \times W + Z$. Злоумышленник не может прочесть сообщение из-за сложности инвертирования матричного умножения при дополнительном наличии вектора ошибки Z .

На приемной стороне получатель, зная по отдельности P, G и S , сначала выполняет умножение $C = C \times (P^{-1})$, затем над вектором C производит декодирование помехоустойчивого кода, удаляя тем самым внесенную случайную ошибку. Полученный после декодирования двоичный вектор M умножается на обратную матрицу S : $M = M \times (S^{-1})$. Сообщение дешифровано. Криптосхема Мак-Элиса приведена на рис. 34.

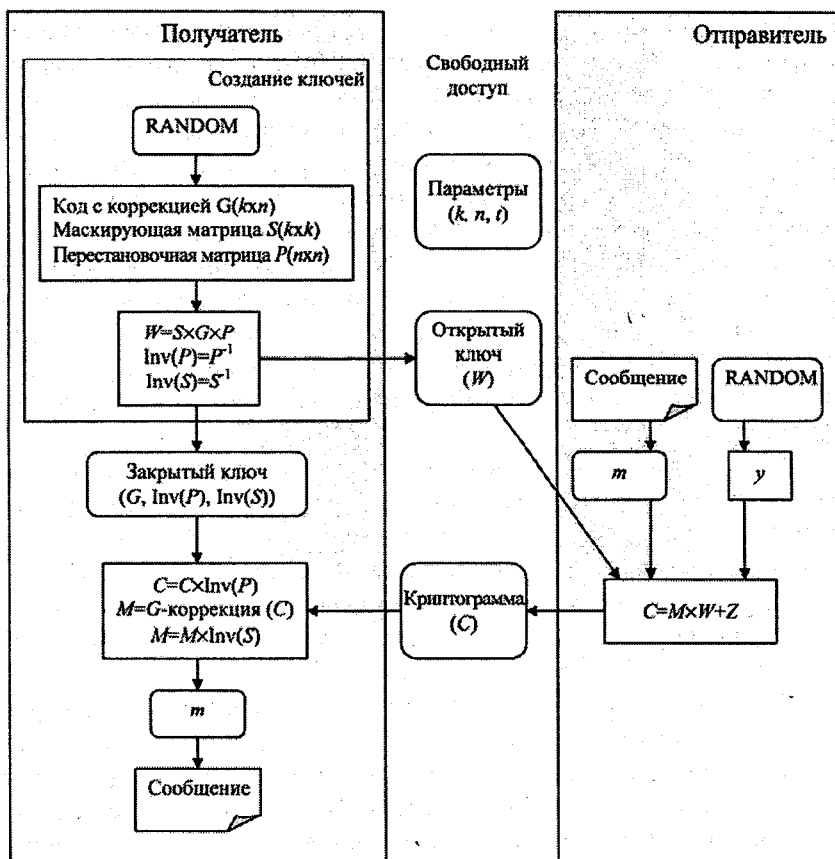


Рис. 34. Криптосистема Мак-Элиса

4. МЕТОДЫ И СРЕДСТВА ОБЕСПЕЧЕНИЯ ПОДЛИННОСТИ ИНФОРМАЦИИ

4.1. Обобщенная система цифровой подписи

Авторы схемы открытого шифрования Диффи и Хеллман показали возможность ее использования для получения *цифровых подписей*. В общем случае цифровая подпись представляет собой некоторое число со специфической структурой, которое допускает с помощью открытого ключа проверку того факта, что оно было выработано для некоторого сообщения с использованием секретного ключа. Для реализации цифровой подписи необходимо выбрать такую одностороннюю функцию с потайным ходом (с секретом) f_z , для которой при всех значениях параметра z область определения функции f_z совпадает с областью ее значений. При этом условии для любого сообщения, которое может быть представлено в виде числа из области определения функции $f_z(x)$, абонент i может сформировать с помощью секретного алгоритма число $S = f_{z_i}^{-1}(M)$. Если сообщение слишком велико, то его можно разбить на части необходимого размера и каждую из них подписать независимо.

Каждый пользователь криптосистемы может по значению S восстановить сообщение M , используя открытый алгоритм шифрования E_{z_i} . Если M представляет собой осмысленное сообщение или может быть сопоставлено с таковым по некоторому заранее оговоренному правилу, то значение S может рассматриваться как цифровая подпись абонента i под сообщением M . Действительно, только владелец секретного алгоритма D_{z_i} мог выработать "открытый" текст S , который с помощью алгоритма E_{z_i} зашифровывается в осмысленную криптограмму M , поскольку лишь абонент i знает способ вычисления $f_{z_i}^{-1}$.

Абонент i может послать абоненту j также секретное сообщение с подписью. Для этого он зашифровывает S с помощью открытого алгоритма E_{z_j} , получая криптограмму $C = E_{z_j}(S)$. Получив зашифрованное сообщение, j -ый абонент расшифровывает его своим

секретным алгоритмом $D_{z_j}(C) = S$, затем число S зашифровывает открытым алгоритмом i -го абонента $E_{z_i}(S) = M$. Таким образом, по полученной криптограмме C абонент j восстанавливает подпись абонента i и исходное сообщение.

Использование протоколов на основе симметричных криптографических методов предполагает, что две стороны *доверяют друг другу*. Криптосистемы с открытым ключом (асимметричные криптосистемы) позволяют реализовать протоколы взаимодействия сторон, которые *не доверяют друг другу*. Системы электронной цифровой подписи являются одним из важнейших примеров такого типа. Для того чтобы цифровая подпись могла эффективно использоваться в реальных деловых взаимоотношениях абонентов, следует придать ей юридическую силу. Для этого необходимо принять соответствующие правовые законы на государственном (или международном) уровне, а обмен открытыми ключами подкреплять обычной юридической процедурой, чтобы обеспечить защиту отказов от открытых ключей.

При формировании электронной подписи под сообщением, отправляемым неким абонентом в той же единой информационной системе, отправитель *подписывает* послание своим секретным ключом. На самом деле пользователь вычисляет контрольную сумму сообщения, шифрует её секретным ключом и приклеивает шифrogramму к сообщению, однако глагол очень удачно вписался в новое значение и теперь неотделим от понятия ЭЦП.

При создании электронной подписи под документом в нее закладывается достаточно информации, чтобы любой получатель мог удостовериться с помощью открытого ключа отправителя, что только он мог подписать, а следовательно, и отправить это сообщение. Но при этом в подписи должно быть недостаточно информации, чтобы извлечь из нее сам ключ отправителя — иначе после первого подписания любой перехвативший письмо мог бы подписывать от имени отправителя свои послания, т.е. технология ЭЦП очень напоминает асимметричный шифр, только наоборот.

4.2. Электронная цифровая подпись RSA

Инверсия асимметричного алгоритма RSA для получения схемы цифровой подписи является, пожалуй, самой простой из всех алгоритмов ЭЦП. Для формирования электронной подписи отправитель выполняет над контрольной суммой документа те же самые действия, что и при шифровании, но использует не открытый ключ получателя, а свой собственный закрытый ключ, т.е. $sign_i = (h_i^d \bmod n)$. Открытый и закрытый ключи просто меняются местами. На приемной стороне получатель возводит подпись в степень открытого ключа e отправителя и получает $(sign_i^e \bmod n) = (h_i^{de} \bmod n) = h_i$.

Если получившееся после возведения в степень значение совпадает с вычисленной независимо на приемной стороне контрольной суммой документа, то проверка считается выполненной, а документ – подписанным. Никто, кроме отправителя, не зная d , не может вычислить такую подпись $sign_i$, чтобы возведение ее в степень открытого ключа e дало требуемую контрольную сумму – это та же самая трудноразрешимая задача, что и асимметричном шифровании RSA. Следовательно, снабдить документ такой подписью $sign_i$ мог только истинный владелец закрытого ключа. Схема ЭЦП RSA приведена на рис. 35.

Использование секретного ключа для цифровой подписи, т.е. раскрытие некоторой информации о нем не приносит ему ущерба. Даже перехватив большое количество подписанных сообщений, злоумышленник не сможет с их помощью вычислить d за полиномиальное время, а применяемой на сегодняшний день длины ключа хватит на несколько десятков лет стойкости.

В такой простой схеме ЭЦП есть на самом деле другая потенциальная уязвимость: если злоумышленник активно переписывается с атакуемым абонентом, то умелым сочетанием шифрования и подписей со стороны абонента он сможет читать сообщения, адресованные атакуемым абонентам всем своим корреспондентам. Например, добившись под каким-либо предлогом подписания абонентом нескольких байт абракадабры, он сможет прочесть одно послание этому абоненту. Произойдет это, если злоумышленник поместил в эту кажущуюся бессмысленной последовательность

байт перехваченный ключ сеанса, зашифрованный открытым ключом: $u = (k_i^e \bmod n)$. Тогда, будучи подписанной, $(u^d \bmod n)$ представляет собой секретный ключ блочного шифра перехваченного послания в открытом виде: $(u^d \bmod n) = (k_i^{ed} \bmod n) = k_i$.

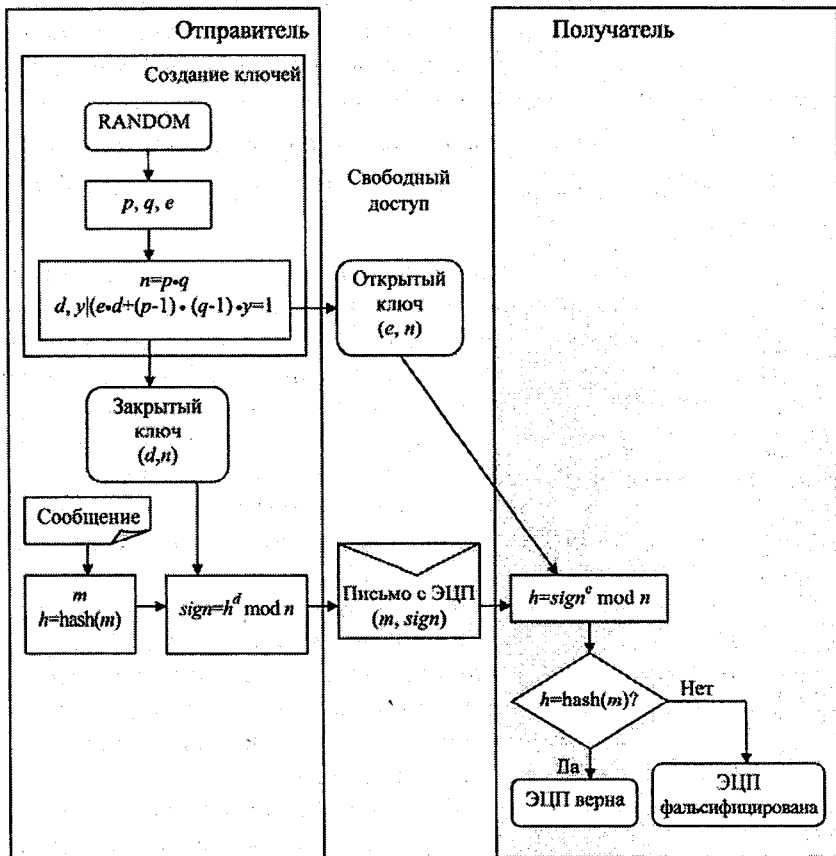


Рис. 35. ЭЦП RSA

Частное решение этой проблемы — никогда не подписывать то, чего не создавал сам. Общее — принять за правило подписывать только контрольную сумму интересующего сообщения, даже если все оно настолько мало, что поместится в один блок преобразования RSA. При этом для подобных целей повсеместно используют-

ся только криптостойкие (необратимые) контрольные суммы — хэш-функции. Они не дают злоумышленнику возможности сформировать документ так, чтобы его контрольная сумма совпала с заранее желаемым значением (u), а следовательно, абонент всегда будет подписывать уникальный битовый вектор, снимая возможность описанной атаки.

4.3. Электронная цифровая подпись Рабина

В цифровой подписи по схеме Рабина отправитель выполняет над контрольной суммой такие действия, которые сделать может только он сам, зная как раскладывается на сомножители p и q его открытый ключ n . Это действие — извлечение квадратного корня в поле натуральных чисел $[0; n-1]$ по модулю n . При отправке абонент вычисляет квадратный корень из контрольной суммы, на принимающей стороне получателю остается только возвести ее в квадрат и проверить, совпало ли полученное значение с контрольной суммой подписанного ею документа. Схема ЭЦП по методу Рабина приведена на рис. 36.

Проблема с этой, казалось бы, очень простой схемой в том, что не всякое число имеет квадратный корень в поле по модулю простого числа. Наличие квадратного корня высоко вероятно, но не обязательно. Для решения этой задачи существуют два способа. Согласно первому в подписываемом значении h резервирует несколько бит и перебирает все возможные их комбинации, добиваясь наличия квадратного корня у сгенерированного таким образом числа. По теории вероятности из 4 попыток хотя бы одно из чисел будет иметь квадратный корень с очень большой вероятностью. На приемной стороне добавленные биты игнорируются. Второй способ носит название модифицированная ЭЦП по схеме Рабина и гарантирует наличие квадратного корня у подписываемого числа.

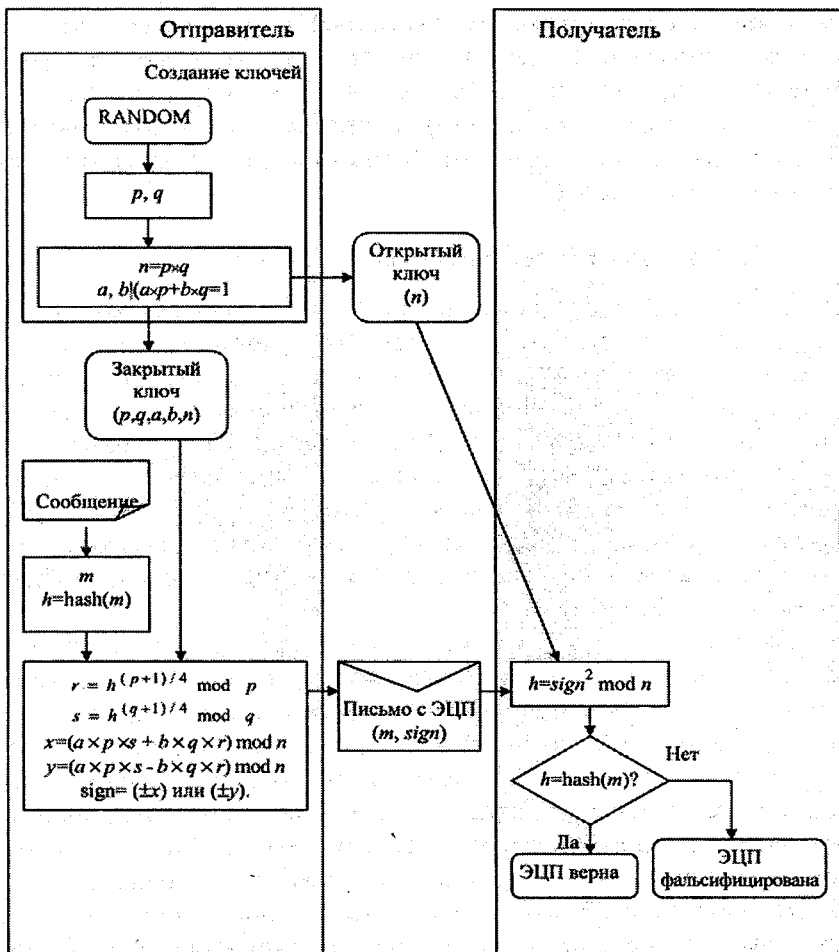


Рис. 36. ЭЦП Рабина

4.4. Электронная цифровая подпись Эль Гамаль

Схема Эль Гамаль является одной из самых распространенных схем ЭЦП. Этому послужило, во-первых, то, что при надлежащей достаточно хорошо проверенной стойкости схема имеет хорошую скорость вычисления. Во-вторых, схема имеет достаточно много модификаций, что в принципе мало свойственно асим-

метричным шифрам. В схеме Эль Гамаль абонент, подписывающий документ доказывает всем желающим проверить подпись, что знает секретный ключ x — степень, в которую был возведен образующий элемент a , чтобы получить открытый ключ b . Явным образом продемонстрировать x , естественно, нельзя, иначе любой другой участник информационного обмена начнет подписывать им документы. Поэтому приходится идти на математические уловки — демонстрировать не само число x , а результат некой математической формулы с участием x . При проверке само число x ни на каком этапе не раскрывается, но проверяющий на основе этой формулы удостоверяется, что отправитель сообщения действительно знает x .

Базовый вариант ЭЦП по схеме Эль Гамаль выглядит следующим образом.

На этапе подписания отправитель:

1. Генерирует случайное число k , уникальное для каждого подписываемого документа, взаимно простое с числом $(p - 1)$.

2. Вычисляет $p = (a^k \bmod p)$.

3. Вычисляет обратный элемент поля к числу k — его обозначают $(k^{-1} \bmod (p-1))$ или $1/k$, в дальнейшем операция «деление на k » равнозначна умножению на $1/k$.

4. Вычисляет $s = ((h - x^*r)/k) \bmod (p-1)$, где h — контрольная сумма подписываемого документа.

Пара числе (r, s) является цифровой подписью для документа, имеющего контрольную сумму h .

На приемной стороне получатель:

1. Вычисляет $u = ((b^*)^x (r^s) \bmod p)$.

2. Вычисляет $v = (a^h \bmod p)$.

3. Проверяет равенство значений $u = v$ (если равенство выполняется, то подпись верна, в противном случае документ сфальсифицирован).

Почему данное равенство является тождеством в случае корректности ЭЦП? Произведем несложные преобразования над u (подразумеваем, что все действия выполняются по модулю числа p):

$$u = ((b^*)^x (r^s)) = ((a^y)^x) * ((a^k)^s) = a^{(ux + ks)} = a^{(ux + hr)} = a^h = v.$$

Как видим, во-первых, получилось тождество, а во-вторых, ни на одном этапе преобразований (а все они доступны проверяющему)

число x не фигурировало в открытом виде — его как бы «экранируют» две величины r и k , причем вторая не известна получателю. Схема ЭЦП Эль Гамаль приведена на рис. 37.

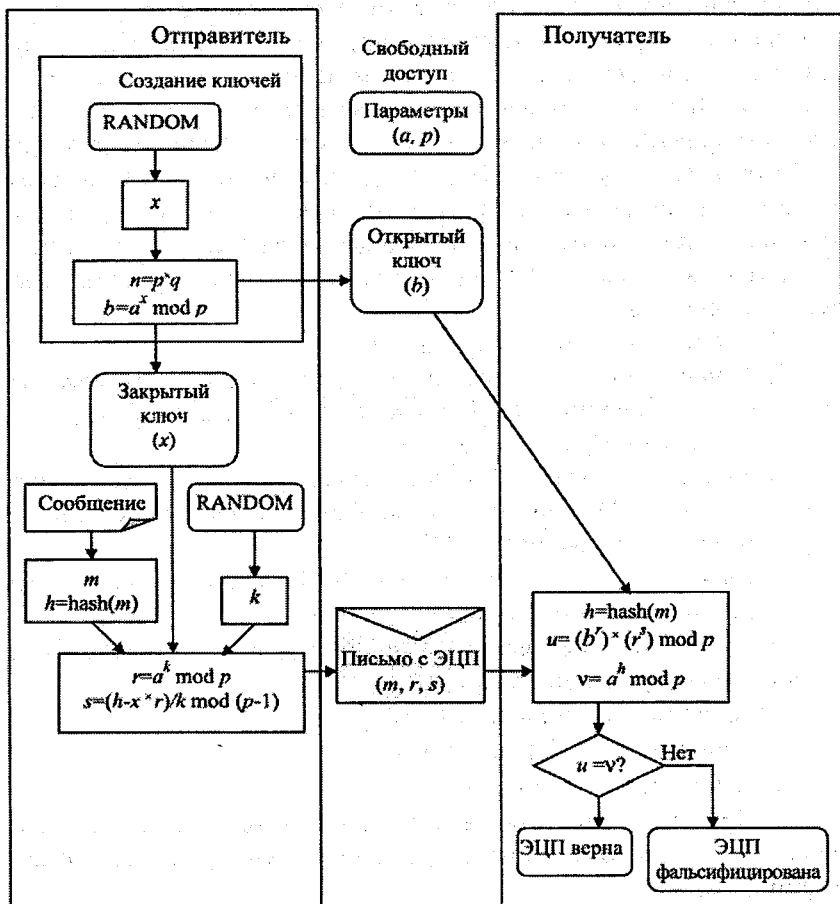


Рис. 37. ЭЦП Эль Гамаль

В качестве особенностей ЭЦП схемы Эль Гамаль необходимо упомянуть требование проверять на приемной стороне неравенство $(r < p)$. Если такой проверки не делать, то злоумышленник сможет подделать подпись, имея перехваченным хотя бы один документ, подписанный легальным пользователем. Правда, по этому

алгоритму значение r получается достаточно большим, что и позволяет защищаться от него сравнением r с простым числом p . Вторым требованием является наличие у числа $(p - 1)$ большого простого делителя. Обычно первоначально выбирается большое простое число q разрядности на один бит меньше требуемой, а затем p вычисляется как $p = 2^x q + 1$ и проверяется на простоту.

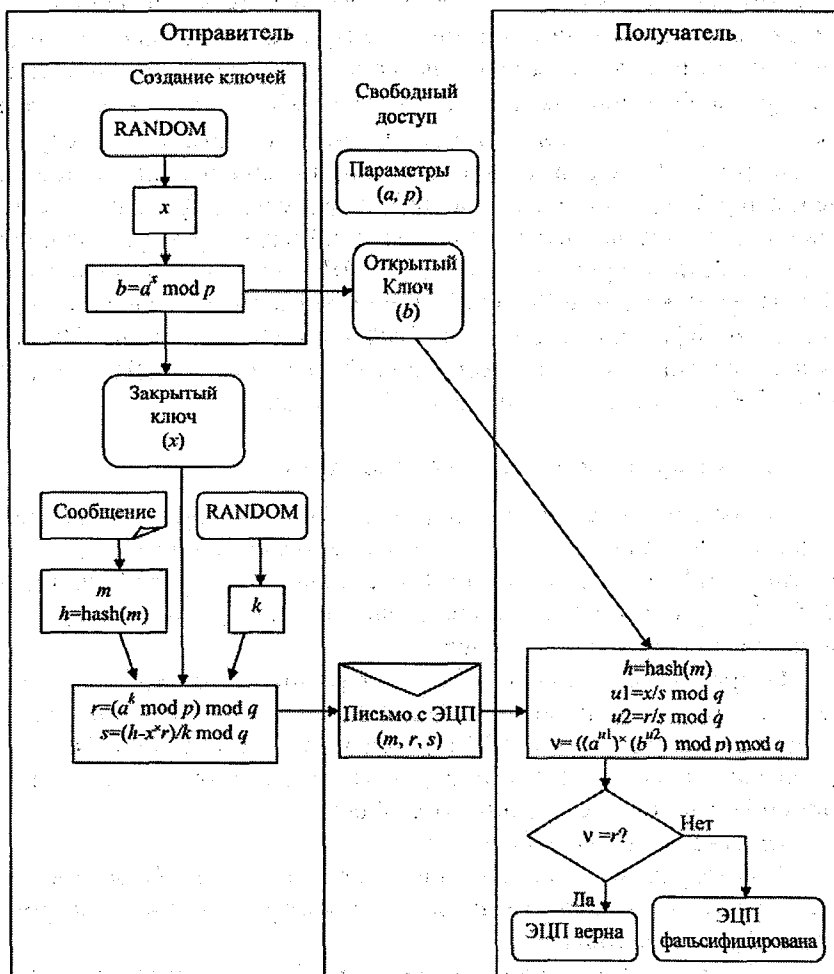


Рис. 38. Алгоритм ЭЦП DSS

Многочисленные модификации алгоритма Эдг Гамаль связаны с несколькими шагами алгоритма. Во-первых, существует возможность при небольшом изменении формул выполнять все вычисления не по модулю p , а по модулю q , являющемуся наибольшим простым делителем p . Во-вторых, формула, вовлекающая секретный ключ и контрольную сумму документа и выглядящая в базовом варианте как $h = k^x s + x^x r$ (это шаг 4 выработки подписи: k перенесено из знаменателя в числитель, а затем слагаемые поменяны местами), является не единственным возможным соотношением. Другие формулы имеют вид: $h = x^x s + k^x r$, $s = x^x r + k^x h$, $s = x^x h + k^x r$, $r = x^x s + k^x h$, $r = x^x h + k^x s$. Естественно, в каждом из этих случаев и формула проверки меняет свой вид, но основная идея остается неизменной. Третий «пласт» модификации схемы Эль Гамаль связан с возможностью переноса вычислений в группу, образованную эллиптическими кривыми. Стандарт США электронной цифровой DSS (Digital Signature Standard), принятый в 1992 г., является одной из модификаций схемы Эль Гамаль и приведен на рис. 38. Государственный стандарт Российской Федерации на ЭЦП ГОСТ Р 34.10-2001 также основан на схеме Эль Гамаль.

4.5. Алгоритм цифровой подписи DSA

Алгоритм цифровой подписи DSA (Digital Signature Algorithm) предложен в 1991 г. в национальном институте стандартов США для использования в стандарте цифровой подписи DSS. Алгоритм DSA является развитием алгоритмов цифровой подписи Эль Гамала и К.Шнорра.

Отправитель и получатель электронного документа используют при вычислении большие целые числа: g и p – простые числа, L бит каждое (от 512 до 1024); q – простое число длиной 160 бит (делитель числа $(p-1)$). Числа g , p , q являются открытыми и могут быть общими для всех пользователей сети.

На этапе подписания отправитель:

1. Выбирает случайное целое число x , $1 < x < q$. Число x является секретным ключом отправителя для формирования электронной цифровой подписи.

2. Вычисляет значение $y = g^x \bmod p$. Число y является открытым ключом для проверки подписи отправителя. Число y передается всем получателям документов.

3. Для того чтобы подписать документ m , отправитель хэширует его в целое хэш-значение $h = \text{hash}(m)$, $1 < h < q$.

4. Генерирует случайное целое число k , $1 < k < q$, и вычисляет число $r = (g^k \bmod p) \bmod q$.

5. Отправитель вычисляет с помощью секретного ключа x целое число $s = \left(\frac{m + r \times x}{k} \right) \bmod q$.

Пара чисел r и s образует цифровую подпись (r, s) под документом m . Таким образом, подписанное сообщение представляет собой тройку чисел (m, r, s) .

На приемной стороне получатель подписанного сообщения (m, r, s) :

1. Проверяет выполнение условий $0 < r < q$, $0 < s < q$ и отвергает подпись, если хотя бы одно из этих условий не выполнено.

2. Вычисляет:

– значение $w = (1/s) \bmod q$,

– хэш-значение $h = \text{hash}(m)$,

– числа $u_1 = (h \times w) \bmod q$, $u_2 = (r \times w) \bmod q$.

3. С помощью открытого ключа y вычисляет значение $v = ((g^{u_1} \times y^{u_2}) \bmod p) \bmod q$ и проверяет выполнение условия $v = r$.

4. Если условие $v = r$ выполняется, тогда подпись (r, s) под документом m признается получателем подлинной.

Схема ЭЦП DSA приведена на рис. 39.

По сравнению с алгоритмом цифровой подписи Эль Гамала алгоритм DSA имеет следующие основные преимущества:

– При любом допустимом уровне стойкости, т.е. при любой паре чисел g и p (от 512 до 1024 бит), числа q , x , r , s имеют длину по 160 бит, сокращая длину подписи до 320 бит.

– Большинство операций с числами k , r , s , x при вычислении подписи производится по модулю числа q длиной 160 бит, что сокращает время вычисления подписи.

– При проверке подписи большинство операций с числами u_1 , u_2 , v , w также производится по модулю числа q длиной 160 бит, что сокращает объем памяти и время вычисления.

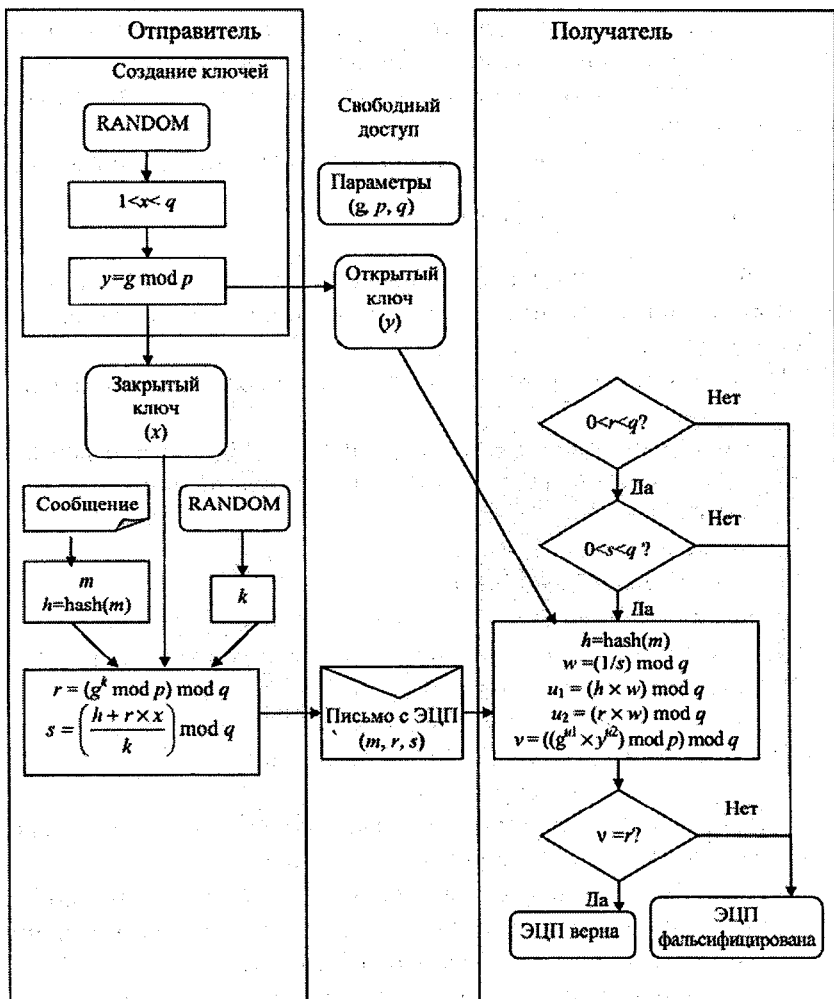


Рис. 39: ЭЦП DSA

Недостатком алгоритма DSA является то, что при подписывании и при проверке подписи приходится выполнять сложные операции деления по модулю q : $s = \left(\frac{m + r \times x}{k} \right) \text{ mod } q$; $w = (1/s) \text{ mod } q$, что не позволяет получать максимальное быстродействие.

Следует отметить, что реальное исполнение алгоритма DSA может быть ускорено с помощью выполнения предварительных вычислений. Заметим, что значение r не зависит от сообщения m и его хэп-значения h . Можно заранее создать строку случайных значений k и затем для каждого из этих значений вычислить значения r . Можно также заранее вычислить обратные значения k^{-1} для каждого из значений k . Затем, при поступлении сообщения m , можно вычислить значение s для данных значений r и k^{-1} . Эти предварительные вычисления значительно ускоряют работу алгоритма DSA.

4.6. Электронная цифровая подпись ГОСТа Р 34.10-2001

Алгоритм Государственного стандарта РФ по ЭЦП (полное название «Процессы формирования и проверки электронной цифровой подписи») является переложением схемы Эль Гамаль в область эллиптических кривых. Общеизвестными параметрами являются сама эллиптическая кривая и точка P на ней, превращающаяся после q -кратного сложения с самой собой в «нулевую» точку кривой. Секретным ключом отправителя d является случайным образом сгенерированное число, а открытым – точка Q , определяемая по формуле ($Q=d \times P$).

На этапе выработки подписи отправитель:

1. Генерирует случайное число k ($0 < k < q$);
2. Вычисляет x -координату точки ($k \times P$), назовем это число r .
3. Вычисляет значение $s = (r \times d + k \times h) \bmod q$, где h – контрольная сумма подписываемого сообщения.

Подписью является пара чисел (r, s)

На этапе проверки выполняются следующие действия:

1. Вычисляются значения ($z_1 = (s \times (h^{-1})) \bmod q$ и ($z_2 = ((-r) \times (h^{-1})) \bmod q$).
2. Определяется x -координата точки ($z_1 \times P + z_2 \times Q$), назовем это число R .
3. Проверяется равенство ($r = R$) (если оно верно, подпись корректна).

Разрядность числа q фиксируется Стандартом в 255 бит. Это означает, что при заданной стойкости открытый и закрытый ключи имеют гораздо меньшую длину и требуют при вычислениях меньших затрат ресурсов. Все это достоинства эллиптических кривых. Схема ЭЦП по ГОСТу Р 34.10-2001 приведена на рис. 40.

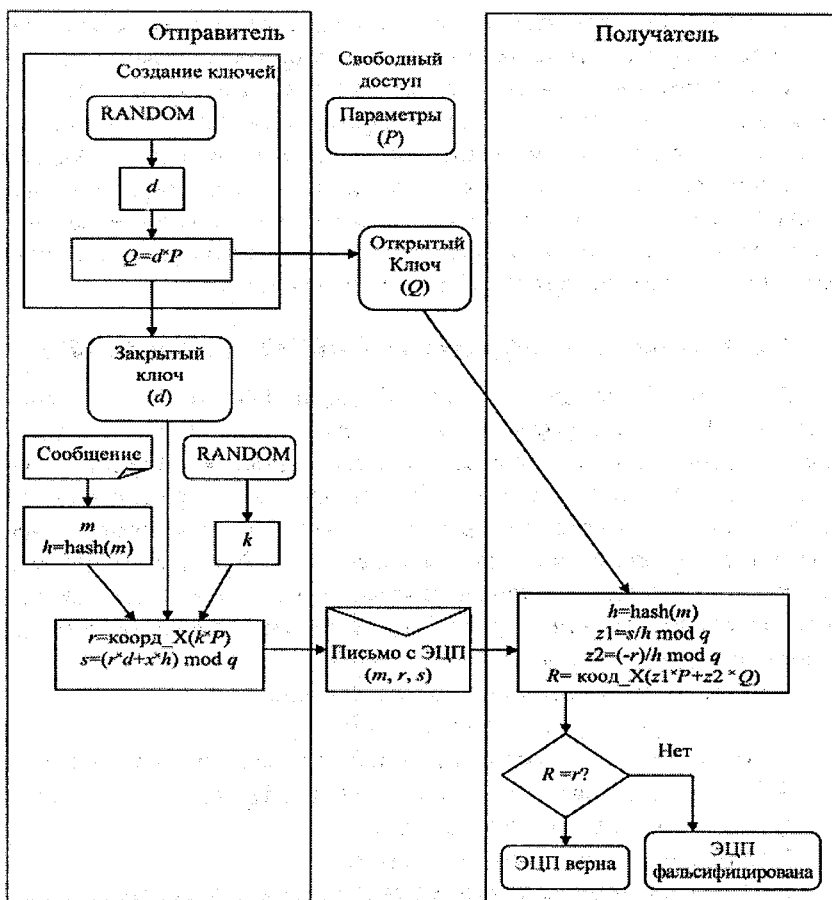


Рис. 40. ЭЦП по ГОСТу Р 34.10-2001

5. Хэш-функции

Хэш-функция предназначена для сжатия подписываемого документа M до нескольких десятков или сотен бит. Хэш-функция h принимает в качестве аргумента сообщение (документ) M произвольной длины и возвращает хэш-значение $h(M) = H$ фиксированной длины. Обычно хэшированная информация является сжатым двоичным представлением основного сообщения произвольной длины. Следует отметить, что значение хэш-функции $h(M)$ сложным образом зависит от документа M и не позволяет восстановить сам документ M .

Хэш-функция должна удовлетворять целому ряду условий:

- хэш-функция должна быть чувствительна к всевозможным изменениям в тексте M , таким как вставки, выбросы, перестановки и т.п.;
- хэш-функция должна обладать свойством необратимости, т.е. задача подбора документа M' , который обладал бы требуемым значением хэш-функции, должна быть вычислительно неразрешима;
- вероятность того, что значения хэш-функций двух различных документов (вне зависимости от их длин) совпадут, должна быть ничтожно мала.

Большинство хэш-функций строится на основе однонаправленной функции $f(\bullet)$, которая образует выходное значение длиной n при задании двух входных значений длиной n . Этими входами являются блок исходного текста M и хэш-значение H_{i-1} предыдущего блока текста (рис.41):

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

Хэш-значение, вычисляемое при вводе последнего блока текста, становится хэш-значением всего сообщения M .

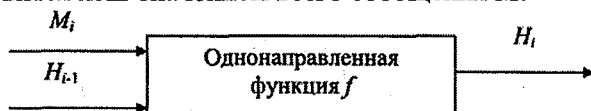


Рис.41. Построение однонаправленной хэш-функции

В результате однонаправленная хэш-функция всегда формирует выход фиксированной длины n (независимо от длины входного текста).

5.1. Однонаправленные хэш-функции на основе симметричных блочных алгоритмов

Однонаправленную хэш-функцию можно построить, используя симметричный блочный алгоритм. Наиболее очевидный подход состоит в том, чтобы шифровать сообщение M посредством блочного алгоритма.

Действительно, если учесть, что любой блочный шифр имеет два входа – открытый текст, то обнаружится, что это идеальный претендент на ядро хэш-функции. Тем более, необходимо отме-

титель тот факт, что блочные шифры необратимы по ключу шифрования, т.е., если порции хэшируемых данных подавать именно на вход «ключ» шифра, то автоматически выполняется условие необратимости хэширования.

В тех случаях, когда требуемая разрядность хэш-функции n совпадает с размером блока шифра, алгоритм сжимающей функции становится предельно простым. Величина $(k-n)$ полагается равной длине ключа блочного шифра, т.е. k равняется сумме размера блока и длины ключа шифра. Существует три основных схемы превращения блочного шифра в сжимающую функцию хэш-алгоритма. Все они имеют примерно равную стойкость и приведены на рис. 42.

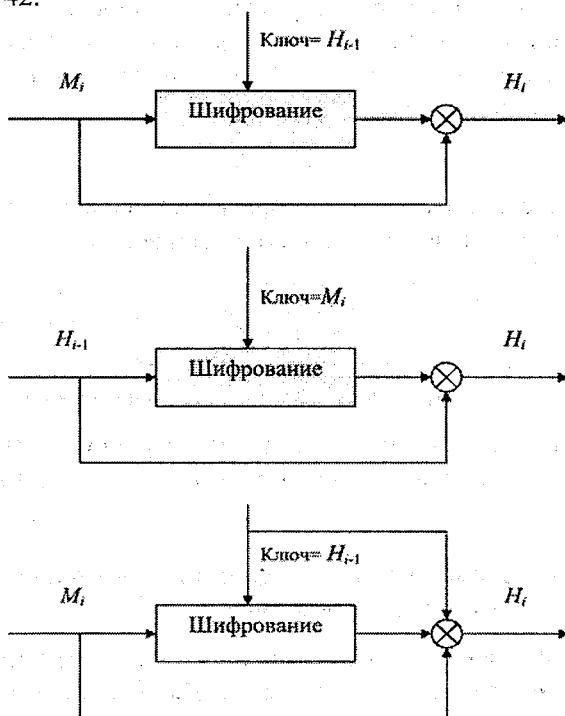


Рис. 42. Хэш-функции на блочных шифрах одинарной размерности

Для тех случаев, когда результат хэширования превосходит размер блока шифра ровно в два раза, также предложено множест-

во различных типовых схем. Одной из таких схем является алгоритм TandemDM (Tandem Davis-Meyer), предложенный Лаи и Мессэй на основе схемы одинарной размерности Дэвиса-Майера и приведен на рис. 43.

Основным недостатком хэш-функций на основе блочных шифров является невысокая скорость их работы.

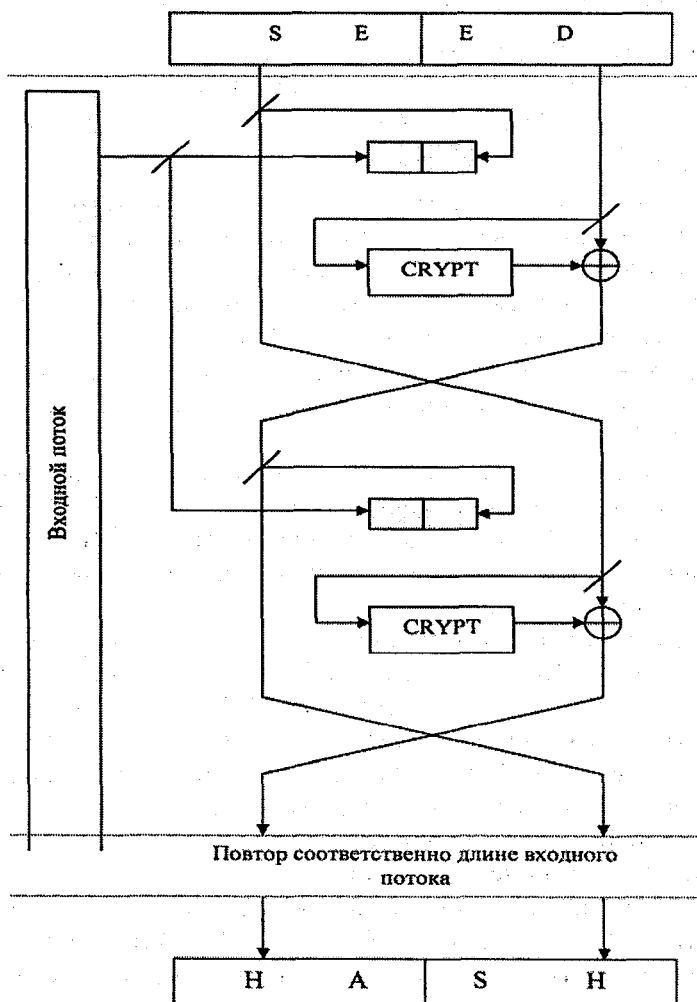


Рис. 43. Алгоритм TandemDM

5.2. Самостоятельные хэш-алгоритмы

5.2.1. Алгоритм MD5

Алгоритм MD5 (Message Digest №5) разработан Рональдом Ривестом на основе своего же предыдущего алгоритма MD4. Алгоритм использует четыре многократно повторяющиеся преобразования над тремя 32-битными величинами X , Y и Z :

$$f(X, Y, Z) = (X \text{ AND } Y) \text{ OR } (\text{NOT } X) \text{ AND } Z$$

$$g(X, Y, Z) = (X \text{ AND } Z) \text{ OR } (Y \text{ AND } (\text{NOT } Z))$$

$$h(X, Y, Z) = X \text{ XOR } Y \text{ XOR } Z$$

$$k(X, Y, Z) = (Y \text{ XOR } (X \text{ OR } (\text{NOT } Z))).$$

В этом алгоритме размер хэш-кода равен 128 битам.

После ряда начальных действий MD5 разбивает текст на блоки длиной 512 битов, которые, в свою очередь, делятся на 16 подблоков по 32 бита. Выходом алгоритма являются 4 блока по 32 бита, конкатенация которых образует 128-битовый хэш-код.

Сначала текст дополняется таким образом, чтобы длина получаемого текста, выраженная в битах, стала на 64 меньше числа, кратного 512.

Дополнение осуществляется приписыванием к концу сообщения единицы и затем необходимого числа нулей (в бинарном представлении). Затем к тексту приписывается 64-битовое представление длины исходного сообщения. Таким образом, получается текст, длина которого кратна 512 битам.

На вход алгоритма поступает входной поток данных, хеш которого необходимо найти. Длина сообщения может быть любой (в том числе нулевой). Запишем длину сообщения в L . Это число целое и неотрицательное. Кратность каким-либо числам необязательна. После поступления данных идёт процесс подготовки потока к вычислениям. Схема работы алгоритма MD5 приведена на рис. 44.

Ниже приведены 5 шагов алгоритма:

Шаг 1. Выравнивание потока

Входные данные выравниваются так, чтобы их размер был сравним с 448 по модулю 512 ($L' = 512 \times N + 448$). Сначала дописывают единичный бит в конец потока, затем необходимое число

нулевых бит (выравнивание происходит, даже если длина уже конгруэнтна – сравнима с 448).

Шаг 2. Добавление длины сообщения

В оставшиеся 64 бита дописывают 64-битное представление длины данных (количество бит в сообщении) до выравнивания. Если длина превосходит $2^{64} - 1$, то дописывают только младшие биты. После этого длина потока станет кратной 512. Вычисления будут основываться на представлении этого потока данных в виде массива слов по 512 бит.

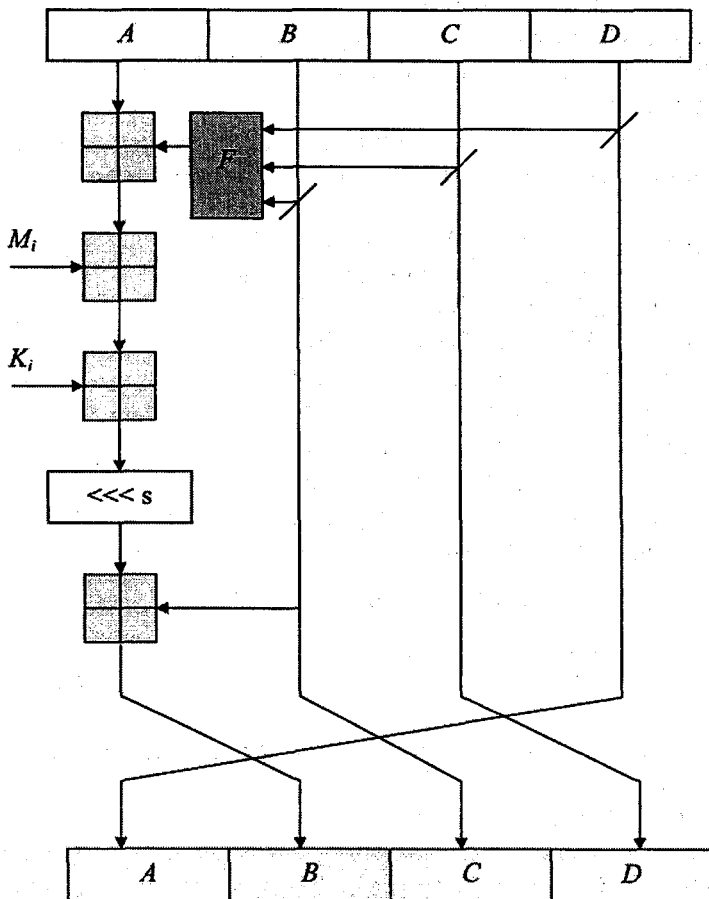


Рис. 44. Схема работы алгоритма MD5

Шаг 3. Инициализация буфера

Для вычислений инициализируются 4 переменных размером по 32 бита и задаются начальные значения шестнадцатеричными числами:

A = 01 23 45 67;

B = 89 AB CD EF;

C = FE DC BA 98;

D = 76 54 32 10.

В этих переменных будут храниться результаты промежуточных вычислений. Начальное состояние ABCD называется инициализирующим вектором.

Определим ещё функции и константы, которые необходимы для вычислений:

– потребуются 4 функции для четырёх раундов. Введём функции от трёх параметров – слов, результатом также будет слово.

1 раунд $f(X \text{ AND } Y) \text{ OR } (\text{NOT } X) \text{ AND } Z$

2 раунд $g(X \text{ AND } Z) \text{ OR } (Y \text{ AND } (\text{NOT } Z))$

3 раунд $h(X \text{ XOR } Y \text{ XOR } Z)$

4 раунд $k(Y \text{ XOR } (X \text{ OR } (\text{NOT } Z)))$;

– определим таблицу констант $T[1..64]$ – 64-элементная таблица данных, построенная следующим образом: $T[i] = \text{int}(4294967296 \times |\sin(i)|)$, где $4294967296 = 2^{32}$;

– выровненные данные разбиваются на блоки (слова) по 32 бита, и каждый блок проходит 4 раунда из 16 операторов. Все операторы однотипны и имеют вид: $[abcd \ k \ s \ i]$, определяемый как

$$a = b + ((a + F(b, c, d) + X[k] + T[i]) << s),$$

где X – блок данных. $X[k] = M[n \times 16 + k]$;

k – номер 32-битного слова из n -го 512-битного блока сообщения;

s – циклический сдвиг влево на s бит полученного 32-битного аргумента.

Шаг 4. Вычисление в цикле

Заносим в блок данных элемент n из массива. Сохраняются значения A, B, C и D, оставшиеся после операций над предыдущими блоками (или их начальные значения, если блок первый).

AA = A

BB = B

CC = C

DD = D

Раунд 1

```
/*[abcd k s i] a = b + ((a + F(b,c,d) + X[k] + T[i]) <<< s). */  
[ABCD 0 7 1][DABC 1 12 2][CDAB 2 17 3][BCDA 3 22 4]  
[ABCD 4 7 5][DABC 5 12 6][CDAB 6 17 7][BCDA 7 22 8]  
[ABCD 8 7 9][DABC 9 12 10][CDAB 10 17 11][BCDA 11 22 12]  
[ABCD 12 7 13][DABC 13 12 14][CDAB 14 17 15][BCDA 15 22 16]
```

Раунд 2

```
/*[abcd k s i] a = b + ((a + G(b,c,d) + X[k] + T[i]) <<< s). */  
[ABCD 1 5 17][DABC 6 9 18][CDAB 11 14 19][BCDA 0 20 20]  
[ABCD 5 5 21][DABC 10 9 22][CDAB 15 14 23][BCDA 4 20 24]  
[ABCD 9 5 25][DABC 14 9 26][CDAB 3 14 27][BCDA 8 20 28]  
[ABCD 13 5 29][DABC 2 9 30][CDAB 7 14 31][BCDA 12 20 32]
```

Раунд 3

```
/*[abcd k s i] a = b + ((a + H(b,c,d) + X[k] + T[i]) <<< s). */  
[ABCD 5 4 33][DABC 8 11 34][CDAB 11 16 35][BCDA 14 23 36]  
[ABCD 1 4 37][DABC 4 11 38][CDAB 7 16 39][BCDA 10 23 40]  
[ABCD 13 4 41][DABC 0 11 42][CDAB 3 16 43][BCDA 6 23 44]  
[ABCD 9 4 45][DABC 12 11 46][CDAB 15 16 47][BCDA 2 23 48]
```

Раунд 4

```
/*[abcd k s i] a = b + ((a + I(b,c,d) + X[k] + T[i]) <<< s). */  
[ABCD 0 6 49][DABC 7 10 50][CDAB 14 15 51][BCDA 5 21 52]  
[ABCD 12 6 53][DABC 3 10 54][CDAB 10 15 55][BCDA 1 21 56]  
[ABCD 8 6 57][DABC 15 10 58][CDAB 6 15 59][BCDA 13 21 60]  
[ABCD 4 6 61][DABC 11 10 62][CDAB 2 15 63][BCDA 9 21 64]
```

Суммируем с результатом предыдущего цикла:

A = AA + A

B = BB + B

C = CC + C

D = DD + D

После окончания цикла необходимо проверить, есть ли ещё блоки для вычислений. Если да, то изменяем номер элемента массива ($n++$) и переходим в начало цикла.

Шаг 5. Результат вычислений

Результат вычислений находится в буфере ABCD, это и есть хеш. Если вывести слова в обратном порядке DCBA, то получим MD5 хеш.

5.2.2. Алгоритм SHA-1

SHA-1 реализует хеш-функцию, построенную на идее функции сжатия. Входами функции сжатия являются блок сообщения длиной 512 бит и выход предыдущего блока сообщения. Выход представляет собой значение всех хеш-блоков до этого момента. Иными словами, хеш блока M_i равен $h_i = f(M_i, h_{i-1})$. Хеш-значением всего сообщения является выход последнего блока.

Инициализация

Исходное сообщение разбивается на блоки по 512 бит в каждом. Последний блок дополняется до длины, кратной 512 бит. Сначала добавляется «1» а потом нули, чтобы длина блока стала равной ($512 - 64 = 448$) бит. В оставшиеся 64 бита записывается длина исходного сообщения в битах. Если последний блок имеет длину более 448, но менее 512 бит, дополнение выполняется следующим образом: сначала добавляется «1», затем «0» вплоть до конца 512-битного блока; после этого создается ещё один 512-битный блок, который заполняется вплоть до 448 бит нулями, после чего в оставшиеся 64 бита записывается длина исходного сообщения в битах. Дополнение последнего блока осуществляется всегда, даже если сообщение уже имеет нужную длину.

Инициализируются пять 32-битовых переменных.

$A = a = 0 \times 67452301$

$B = b = 0 \times \text{EFCDAB89}$

$C = c = 0 \times 98BADCFE$

$D = d = 0 \times 10325476$

$E = e = 0 \times \text{C3D2E1F0}$

Определяются четыре нелинейные операции и четыре константы, представленные в табл. 3.

Таблица 3

Нелинейные операции и константы, применяемые
на этапе инициализации алгоритма SHA-1

$F(m, l, k) = (m \text{ AND } l) \text{ OR } (k \text{ AND } (\text{NOT } m))$	$K_t = 0 \times 5A827999$	$0 \leq t \leq 19$
$F(m, l, k) = (m \text{ XOR } l \text{ XOR } k)$	$K_t = 0 \times 6ED9EBA1$	$20 \leq t \leq 39$
$F(m, l, k) = (m \text{ AND } l) \text{ OR } (m \text{ AND } k) \text{ OR } (l \text{ AND } k)$	$K_t = 0 \times 8F1BBCDC$	$40 \leq t \leq 59$
$F(m, l, k) = (m \text{ XOR } l \text{ XOR } k)$	$K_t = 0 \times CA62C1D6$	$60 \leq t \leq 79$

Главный цикл

Главный цикл итеративно обрабатывает каждый 512-битный блок. Итерация состоит из четырех этапов по двадцать операций в каждом. Блок сообщения преобразуется из 16 32-битовых слов M_t в 80 32-битовых слов W_t по следующему правилу:

$$W_t = M_t, \text{ при } 0 \leq t \leq 15$$

$$W_t = (W_{t-3} \otimes W_{t-8} \otimes W_{t-14} \otimes W_{t-16}) \ll 1, \text{ при } 16 \leq t \leq 79.$$

Здесь $\ll$ – циклический сдвиг влево

для t от 0 до 79

$$\text{temp} = (a \ll 5) + F_t(b, c, d) + e + W_t + K_t$$

$$e = d$$

$$d = c$$

$$c = b \ll 30$$

$$b = a$$

$$a = \text{temp}$$

После этого a, b, c, d, e прибавляются к A, B, C, D, E соответственно. Начинается следующая итерация.

Итоговым значением будет объединение пяти 32-битовых слов в одно 160-битное хеш-значение.

Псевдокод SHA-1

Псевдокод алгоритма SHA-1 следующий (все используемые переменные 32 бита):

Инициализация переменных:

$$h_0 = 0 \times 67452301$$

$$h_1 = 0 \times \text{EFCDAB89}$$

$$h_2 = 0 \times 98BADCFE$$

$$h_3 = 0 \times 10325476$$

$h4 = 0 \times C3D2E1F0$

Предварительная обработка:

Присоединяем бит «1» к сообщению.

Присоединяем k битов «0», где k наименьшее число ≥ 0 такое, что длина получившегося сообщения (в битах) сравнима по модулю 512 с 448).

Добавляем длину исходного сообщения (до предварительной обработки) как целое 64-битное Big-endian число, в *битах*.

В процессе сообщение разбивается последовательно по 512 бит:

for перебираем все такие части

разбиваем этот кусок на 16 частей, слов по 32-бита $w[i]$, $0 \leq i \leq 15$

16 слов по 32-бита дополняются до 80 32-битовых слов:

for i from 16 to 79

$w[i] = (w[i-3] \text{ xor } w[i-8] \text{ xor } w[i-14] \text{ xor } w[i-16])$ циклический

сдвиг 1

Инициализация хеш-значений этой части:

$a = h0$

$b = h1$

$c = h2$

$d = h3$

$e = h4$

Основной цикл:

for i from 0 to 79

if $0 \leq i \leq 19$ then

$f = (b \text{ and } c) \text{ or } ((\text{not } b) \text{ and } d)$

$k = 0 \times 5A827999$

else if $20 \leq i \leq 39$

$f = b \text{ xor } c \text{ xor } d$

$k = 0 \times 6ED9EBA1$

else if $40 \leq i \leq 59$

$f = (b \text{ and } c) \text{ or } (b \text{ and } d) \text{ or } (c \text{ and } d)$

$k = 0 \times 8F1BBCDC$

else if $60 \leq i \leq 79$

$f = b \text{ xor } c \text{ xor } d$

$k = 0 \times \text{CA62C1D6}$

$\text{temp} = (\text{a leftrotate } 5) + f + e + k + w[i]$

$e = d$

$d = c$

$c = b \text{ leftrotate } 30$

$b = a$

$a = \text{temp}$

Добавляем хеш-значение этой части к результату:

$h0 = h0 + a$

$h1 = h1 + b$

$h2 = h2 + c$

$h3 = h3 + d$

$h4 = h4 + e$

Итоговое хеш-значение:

$\text{digest} = \text{hash} = h0 \text{ append } h1 \text{ append } h2 \text{ append } h3 \text{ append } h4$

Примеры

Ниже приведены примеры хешей SHA-1. Для всех сообщений подразумевается использование кодировки ASCII

Хеш панграммы на русском:

SHA-1 («В чащах юга жил бы цитрус? Да, но фальшивый экземпляр!»)

$= 9e32295f \ 8225803b \ b6d5fdfc \ c0674616 \ a4413c1b$

Криптоанализ

Криптоанализ хеш-функций направлен на исследование уязвимости к различного вида атакам. Основные из них:

нахождение коллизий – ситуация, когда двум различным исходным сообщениям соответствует одно и то же хеш-значение.

нахождение прообраза – исходного сообщения – по его хешу.

5.2.3. Российский стандарт хэш-функции

ГОСТ Р 34.11-94

Дата введения стандарта: 01 января 1995 г.

Размер хэша: 256 бит.

Размер блока входных данных: 256 бит.

Для описания алгоритма хэширования будем использовать следующие обозначения:

$\{0\}^j$ – блок длиной j бит заполненный нулями.

$|M|$ – длина блока M в битах по модулю 2^{256} .

$\parallel$ – слияние (конкатенация) двух блоков в один.

Основой описываемой хэш-функции является шаговая функция хэширования

$$H_{\text{out}} = f(H_{\text{in}}, m),$$

где $H_{\text{out}}, H_{\text{in}}, m$ – блоки длины 256 бит.

Входное сообщение M разделяется на блоки $m_n, m_{n-1}, m_{n-2}, \dots, m_1$ по 256 бит. В случае если размер последнего блока m_n меньше 256 бит, то к нему приписываются слева нули для достижения заданной длины блока.

Каждый блок сообщения, начиная с первого, подаётся на шаговую функцию для вычисления промежуточного значения хэш-функции:

$$H_{i+1} = f(H_i, m_i).$$

Значение H_1 можно выбрать произвольным. После вычисления H_{n+1} конечное значение хэш-функции получают следующим образом:

$$H_{i+2} = f(H_{n+1}, L),$$

где L – длина сообщения M в битах по модулю 2^{256} .

$$h = f(H_{n+2}, K),$$

где K – контрольная сумма сообщения M : $m_1 + m_2 + m_3 + \dots + m_n$;

h – значение хэш-функции сообщения M .

Схема образования значение хэш-функции h сообщения M приведена на рис. 45.

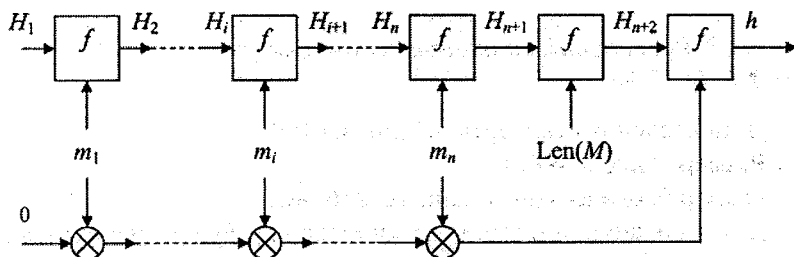


Рис. 45. Схема образования хэш-функции

Алгоритм:

1. Инициализация:

1. $h := \text{initial}$ – начальное значение хэш-функции, т.е. – 256 битовый вектор, определяется пользователем.
2. $\Sigma := 0$ – контрольная сумма.
3. $L := 0$ – длина сообщения.

2. Функция сжатия внутренних итераций: для $i = 1 \dots n-1$ выполняем следующее (пока $|M| > 256$):

1. $h := H(h, m_i)$ – итерация метода последовательного хэширования.
2. $L := L + 256$ – итерация вычисления длины сообщения.
3. $\Sigma := \Sigma + m_i$ – итерация вычисления контрольной суммы.

3. Функция сжатия финальной итерации:

1. $L := L + \lceil m_n \rceil$ – вычисление полной длины сообщения.
2. $m_n := 0^{256 - \lceil m_n \rceil} \| m_n$ – набивка последнего блока.
3. $\Sigma := \Sigma + m_n$ – вычисление контрольной суммы сообщения.
4. $h := H(h, m_n)$
5. $h := H(h, L)$ – MD-усиление.
6. $h := H(h, \Sigma)$.

4. Выход. Значением хэш-функции является h .

Замечание: так как длина сообщения участвует в хэшировании, то нет необходимости указывать в передаваемом сообщении количество добавленных нулей к блоку m_n .

Алгоритм вычисления шаговой функции хэширования

Шаговая функция хэширования H отображает два блока длиной 256 бит в один блок длиной 256 бит: $H_{\text{out}} = f(H_{\text{in}}, m)$ и состоит из трех частей (рис. 46):

- генерирование ключей K_1, K_2, K_3, K_4 ;
- шифрующее преобразование – шифрование H_{in} с использованием ключей K_1, K_2, K_3, K_4 ;
- перемешивающее преобразование результата шифрования.

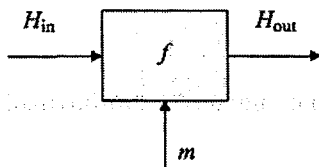


Рис. 46. Шаговая функция хэширования

Генерация ключей

В алгоритме генерации ключей используются:

1. Два преобразования блоков длиной 256 бит
 - преобразование

$$A(Y) = A(y_4 || y_3 || y_2 || y_1) = (y_1 \otimes y_2) || y_4 || y_3 || y_2$$

где y_4, y_3, y_2, y_1 – подблоки блока Y длины 64 бит

- преобразование

$$P(Y) = P(y_{32} || y_{31} || \dots || y_1) = (y_{\varphi(32)} || y_{\varphi(31)} || \dots || y_{\varphi(1)},$$

где $\varphi(i+1+4(k-1)) = 8i+k, i=0, \dots, 3, k=1, \dots, 8,$

$y_{32}, y_{31}, \dots, y_1$ – подблоки блока Y длины 8 бит

2. Три константы:

$$C_2 = 0$$

$$C_3$$

$$0 \times \text{ff00ffff000000ffff0000ff00ffff0000ff00ff00ff00ffff00ff00ff00ff00} =$$

$$C_4 = 0$$

Алгоритм:

$$U := H_{\text{in}}, V := m, W := U \otimes V, K_1 = P(W)$$

Для $j = 2, 3, 4$ выполняем следующее:

$$U := A(U) \otimes C_j, V := A(V), W := U \otimes V, K_j = P(W)$$

Шифрующее преобразование

После генерирования ключей происходит шифрование H_{in} по ГОСТу 28147-89 в режиме простой замены на ключах K_i (для $i := 1, 2, 3, 4$), процедуру шифрования обозначим через E (Примечание: функция шифрования E по ГОСТу 28147 шифрует 64-битные данные 256-битным ключом). Для шифрования H_{in} разделяют на четыре блока по 64 бита: $H_{\text{in}} = h_4 || h_3 || h_2 || h_1 ||$ и зашифровывают каждый из блоков:

$$s_1 = E(h_1, K_1);$$

$$s_2 = E(h_2, K_2);$$

$$s_3 = E(h_3, K_3);$$

$$s_4 = E(h_4, K_4).$$

После чего блоки собирают в 256-битный блок: $S = s_4 || s_3 || s_2 || s_1 ||$

Перемешивающее преобразование

На последнем этапе происходит перемешивание H_{in} , S и m с применением регистра сдвига, в результате чего получают H_{out} .

Для описания процесса преобразования сначала необходимо определить функцию Ψ (рис. 47), которая производит элементарное преобразование блока длины 256 бит в блок той же длины:

$$\begin{aligned}\Psi(Y) &= \Psi(y_{16} \| y_{15} \| \dots \| y_2 \| y_1) = \\ &= (y_1 \otimes y_2 \otimes y_3 \otimes y_4 \otimes y_{13} \otimes y_{16}) \| y_{16} \| y_{15} \| \dots \| y_3 \| y_2,\end{aligned}$$

где $y_{16}, y_{15}, \dots, y_2, y_1$ – подблоки блока Y длины 16 бит.

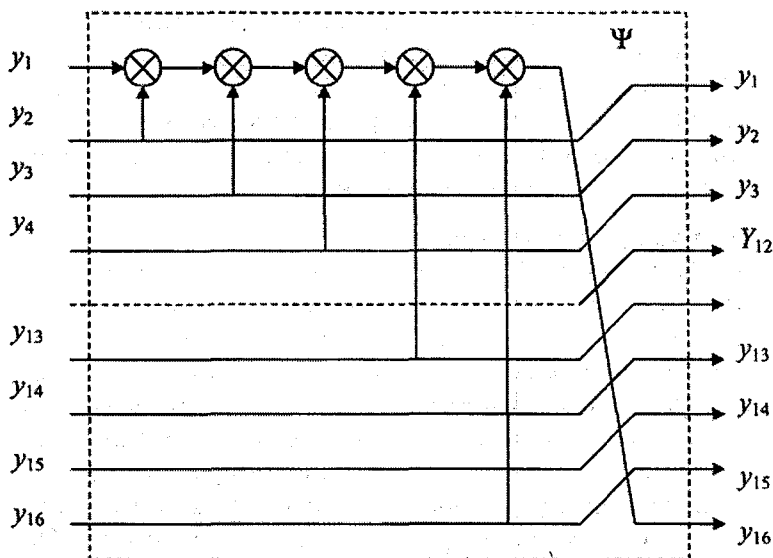


Рис. 47. Элементарное преобразование Ψ

Перемешивающее преобразование имеет вид

$$H_{out} = \Psi^{61}(H_{in} \otimes \Psi(m \otimes \Psi^{12}(S))),$$

где Ψ^i означает i -ю степень преобразования Ψ . Другими словами, преобразование Ψ представляет собой LFSR – линейный сдвиговый регистр с обратной связью, а индекс i указывает на количество раундов LFSR (рис. 48).

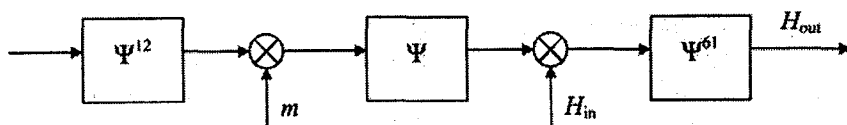


Рис. 48. Перемешивающее преобразование

Ряд отличительных особенностей ГОСТа Р 34.11-94

При обработке блоков используются преобразования по алгоритму «ГОСТ 28147-89».

Обрабатывается блок длиной 256 бит, и выходное значение тоже имеет длину 256 бит.

Применены меры борьбы против поиска коллизий, основанном на неполноте последнего блока.

Обработка блоков происходит по алгоритму шифрования «ГОСТ 28147-89», который содержит преобразования на S-box, что существенно осложняет применение метода дифференциального криптоанализа к поиску коллизий.

Оценка криптостойкости

В 2008 г. командой экспертов из Австрии и Польши была обнаружена техническая уязвимость, сокращающая поиск коллизий в 2^{23} раз. Количество операций, необходимое для нахождения коллизии, таким образом, составляет 2^{105} , что, однако, на данный момент практически не реализуемо. Проведение коллизийной атаки на практике имеет смысл только в случае цифровой подписи документов, причем, если взломщик может изменять неподписанный оригинал.

Функция используется при реализации систем цифровой подписи на базе асимметричного криптоалгоритма по стандарту ГОСТа Р 34.10-2001. Сообщество российских разработчиков систем комплексной защиты информации согласовала используемые в Интернет параметры ГОСТа Р 34.11-94

Примеры хэшей

В качестве шифрующего преобразования в приводимых ниже примерах используется алгоритм ГОСТа 28147-89 в режиме простой замены. При этом заполнение узлов замены (S-box) следующее (табл. 4):

Заполнение узлов замены (S-box)

Номер S-box	Значение															
1	4	10	9	2	13	8	0	14	6	11	1	12	7	15	5	3
2	14	11	4	12	6	13	15	10	2	3	8	1	0	7	5	9
3	5	8	1	13	10	3	4	2	14	15	12	7	6	0	9	11
4	7	13	10	1	0	8	9	15	14	4	6	12	11	2	5	3
5	6	12	7	1	5	15	13	8	4	10	9	14	0	3	11	2
6	4	11	10	0	7	2	1	13	3	6	8	5	9	12	15	14
7	13	11	4	1	3	15	5	9	0	10	14	7	6	8	2	12
8	1	15	13	0	5	7	10	4	9	2	3	14	6	11	8	12

В качестве стартового вектора взято нулевое значение, т.е.
 $H_1 = 0 \times 00000000 \ 00000000 \ 00000000 \ 00000000 \ 00000000 \ 00000000$
 $00000000 \ 00000000$.

Подробный пример из описания стандарта

$M = 0 \times 73657479622032333D6874676E656C202C6567617373656$
 $D2073692073696854$

Так как длина сообщения равна 256 битам, то нет необходимости дописывать нули. Вычисляем $H_2 = f(H_1, M)$:

Генерация ключей

$K_1 = 0 \times 733D2C20 \ 65686573 \ 74746769 \ 79676120 \ 626E7373$
 $20657369 \ 326C6568 \ 33206D54$

$K_2 = 0 \times 110C733D \ 0D166568 \ 130E7474 \ 06417967 \ 1D00626E$
 $161A2065 \ 090D326C \ 4D393320$

$K_3 = 0 \times 80B111F3 \ 730DF216 \ 850013F1 \ C7E1F941 \ 620C1DFF$
 $3ABA E91A \ 3FA109F2 \ F513B239$

$K_4 = 0 \times A0E2804E \ FF1B73F2 \ ECE27A00 \ E7B8C7E1 \ EE1D620C$
 $AC0CC5BA \ A804C05E \ A18B0AEC$

Шифрующее преобразование

$S_1 = 0 \times 42ABBCCE \ 32BC0B1B$

$S_2 = 0 \times 5203EBC8 \ 5D9BCFFD$

$S_3 = 0 \times 8D345899 \ 00FF0E28$

$S_4 = 0 \times E7860419 \ 0D2A562D$

$S = 0 \times E7860419 \ 0D2A562D \ 8D345899 \ 00FF0E28 \ 5203EBC8$
 $5D9BCFFD \ 42ABBCCE \ 32BC0B1B$

Перемешивающее преобразование

$H_2=0 \times \text{CF9A8C65 } 505967\text{A4 } 68\text{A03B8C } 42\text{DE7624 } \text{D99C412A}$
 $883\text{DA687 } 561\text{C7DE3 } 3315\text{C034}$

Вычисляем $H_3=f(H_2, L)$:

$L=0 \times 00000000 \quad 00000000 \quad 00000000 \quad 00000000 \quad 00000000$
 $00000000 \quad 00000000 \quad 00000100$

Генерация ключей

$K_1=0 \times \text{CF68D956 } 9\text{AA09C1C } 8\text{C3B417D } 658\text{C24E3 } 50428833$
 $59\text{DE3D15 } 6776\text{A6C1 } 44248734$

$K_2=0 \times 8\text{FCF68D9 } 809\text{AAC9C } 3\text{C8C3B41 } \text{C7658C24 } \text{BB504288}$
 $2859\text{DE3D } 666676\text{A6 } \text{B3A42487}$

$K_3=0 \times 4\text{E70CF97 } 3\text{C8065A0 } 853\text{C8CC4 } 57389\text{A8C } \text{CABB50BD}$
 $\text{E3D7A6DE } \text{D1996788 } 5\text{CB35B24}$

$K_4=0 \times 584\text{E70CF } 5\text{C53C8065 } 48853\text{C8C } 1657389\text{A } \text{EDCABB50}$
 $78\text{E3D7A6 } \text{EED19867 } 7\text{F5CB35B}$

Шифрующее преобразование

$S = 0 \times 66\text{B70F5E } \text{F163F461 } 468\text{A9528 } 61\text{D60593 } \text{E5EC8A37}$
 $3\text{FD42279 } 3\text{CD1602D } \text{DD783E86}$

Перемешивающее преобразование

$H_3=0 \times 2\text{B6EC233 } 7\text{BC89E4 } 2\text{ABC2692 } 5\text{FEA7285 } \text{DD3848D1}$
 $\text{C6AC997A } 24\text{F74E2B } 09\text{A3AEF7}$

Вычисляем $H_4=f(H_3, \Sigma)$:

$\Sigma=0 \times 7365747962203233\text{D6874676E656C202C6567617373656D20}$
 73692073696854

Генерация ключей

$K_1=0 \times 5817\text{F104 } 0\text{BD45D84 } \text{B6522F27 } 4\text{AF5B00B } \text{A531B57A}$
 $9\text{C8FDFCA } \text{BB1EFCC6 } \text{D7A517A3}$

$K_2=0 \times \text{E82759E0 } \text{C278D95E } 15\text{CC523C } \text{FC72EBB6 } \text{D2C73DA8}$
 $19\text{A6CAC9 } 3\text{E8440F5 } \text{C0DDDB66A}$

$K_3=0 \times 77483\text{AD9 } \text{F7C29CAA } \text{EB06D1D7 } 641\text{BCAD3}$
 $\text{FBC3DAA0 } 7\text{CB555F0 } \text{D4968080 } 0\text{A9E56BC}$

$K_4=0 \times \text{A1157965 } 2\text{D9FBC9C } 088\text{C7CC2 } 46\text{FB3DD2 } 7681\text{ADCB}$
 $\text{FA4ACA06 } 53\text{EFF7D7 } \text{C0748708}$

Шифрующее преобразование

$S=0 \times 2AEBFA76 \ A85FB57D \ 6F164DE9 \ 2951A581 \ C31E7435$
 $4930FD05 \ 1F8A4942 \ 550A582D$

Перемешивающее преобразование

$H_4=0 \times FAFF37A6 \ 15A81669 \ 1CFF3EF8 \ B68CA247 \ E09525F3$
 $9F811983 \ 2EB81975 \ D366C4B1$

Таким образом, **результат хэширования:**

$H=0 \times FAFF37A6 \ 15A81669 \ 1CFF3EF8 \ B68CA247 \ E09525F3$
 $9F811983 \ 2EB81975 \ D366C4B1$

Другие примеры

$M=0 \times 7365747962203035203D206874676E656C2073616820656$
 $7617373656D206C616E696769726F206568742065736F70707553$

$H=0 \times 0852F5623B89DD57AEB4781FE54DF14EEAFBC1350613$
 $763A0D770AA657BA1A47$

$M = \langle \rangle$

$H=0 \times BD298BCFCFAFB398D76E3FC8FA0951679D6B57BD782$
 $AC7Bf13C03C6848A351D89$

$M = \langle \text{ГОСТ Р 34.11-94} \rangle$

$H=0 \times B6A83547677B534FEDB3C68BC7207129F7454928554CF$
 $2847DD7E30165F1CF30$

$M = \langle \text{The quick brown fox jumps over the lazy dog} \rangle$

$H=0 \times 94421F6D370FA1D16BA7AC5E31296529C968047DCA9$
 $BF4258AC59A0C41FAB777$

Малейшее изменение сообщения в подавляющем большинстве случаев приводит к совершенно другому хэшу вследствие лавинного эффекта. К примеру, при изменении dog на cog получится:

$M = \langle \text{The quick brown fox jumps over the lazy cog} \rangle$

$H=0 \times 45C4EE4EE1D25091312135540D6702E6677F7A73B5DA3$
 $1E10B8BB7AADAC4EBA3$

ЗАКЛЮЧЕНИЕ

Выбор криптографических методов защиты для конкретных информационных систем должен быть основан на глубоком анализе их слабых и сильных сторон. Обоснованный выбор тех или иных методов криптографической защиты в общем случае должен опираться на критерии эффективности.

Наиболее простой критерий такой эффективности – вероятность раскрытия ключа или мощность множества ключей. По сути, это то же самое, что и криптостойкость. Для ее численной оценки можно использовать также и сложность раскрытия шифра путем перебора всех ключей.

Однако этот критерий не учитывает других важных требований к криптосистемам:

- невозможность раскрытия или осмысленной модификации информации на основе анализа ее структуры;
- совершенство используемых протоколов защиты;
- минимальный объем используемой ключевой информации;
- минимальная сложность реализации (в количестве машинных операций), ее стоимость;
- высокая оперативность.

Желательно использование некоторых интегральных показателей, учитывающих указанные факторы.

Часто более эффективным при выборе и оценке криптографической системы является использование экспертных оценок и имитационное моделирование.

В любом случае выбранный комплекс криптографических методов должен сочетать как удобство, гибкость и оперативность использования, так и надежную защиту от злоумышленников циркулирующей в информационной системе информации.

ЛИТЕРАТУРА

1. *Diffie W., Hellman M.E.* New directions in cryptography. IEEE transactions on Information Theory IT-22. 1976. 644-654 p.
2. *Rivest R.L., Shamir A., Adleman L.*, Method for obtaining digital signatures and public-key cryptosystems, Commun. ACM, v. 21, №2, 1978, 120-126
3. *Rivest R.L.*, The MD5 message digest algorithm, RFC 1321, April, 1992
4. *Конев И.Р., Беляев А.В.* Информационная безопасность предприятия. – СПб.: БХВ-Петербург, 2003.
4. *Саломеа А.* Криптография с открытым ключом. М., 1996.
5. *Schneier B.* Applied Cryptography. John Wiley & Sons, 1996.
6. *Н.А. Молдовян, А.А. Молдовян, М.А. Еремеев* Криптография: от примитивов к синтезу алгоритмов. – СПб.: БХВ-Петербург, 2004.
7. *Иванов. М.А.* Криптографические методы защиты информации в компьютерных системах и сетях. –М.: КУДИЦ-ОБРАЗ, 2001.
8. *Столингс В.* Криптография и защита сетей. Принципы и практика. 2-е изд. – М.: Вильямс, 2001.

СОДЕРЖАНИЕ

Введение	3
1. Классификация шифров по ключевой информации	6
2. Симметричная криптография. Стандарты блочного шифрования	8
2.1. Аппаратное шифрование DES	9
2.2. Криптоалгоритм TEA	15
2.3. Криптоалгоритм DEA	18
2.4. Криптоалгоритм ГОСТ 28147-89	21
2.5. Алгоритмы конкурса AES	24
2.5.1. MARS	26
2.5.2. RC6	37
2.5.3. Rijndael	41
2.5.4. TwoFish	44
2.5.5. Serpent	48
2.5.6. Итоги конкурса AES	51
3. Системы шифрования с открытым ключом	52
3.1. Основные принципы	52
3.2. Практические схемы асимметричного шифрования	56
3.2.1. Криптоалгоритм RSA	56
3.2.2. Криптоалгоритм Рабина	60
3.2.3. Криптоалгоритм Эль Гамаль	61
3.2.4. Криптоалгоритм Мак-Элиса	64
4. Методы и средства обеспечения подлинности информации	67
4.1. Обобщенная система цифровой подписи	67
4.2. Электронная цифровая подпись RSA	69
4.3. Электронная цифровая подпись Рабина	71
4.4. Электронная цифровая подпись Эль Гамаль	72
4.5. Алгоритм цифровой подписи DSA	76
4.6. Электронная цифровая подпись ГОСТа Р 34.10-2001	79
5. Хэш-функции	80
5.1. Однонаправленные хэш- функции на основе симметричных блочных алгоритмов	81
5.2. Самостоятельные хэш-алгоритмы	84
5.2.1. Алгоритм MD5	84
5.2.2. Алгоритм SHA-1	88
5.2.3. Российский стандарт хэш-функции ГОСТ Р 34.11-94	91
Заключение	100
Литература	101

CONTENTS

Introduction	3
1. Classification of codes by the key information	6
2. Symmetric cryptography. Standards of block encryption	8
2.1. Hardware encryption DES	9
2.2. Encryption algorithm DEA	15
2.3. Encryption algorithm TEA	18
2.4. Encryption algorithm GOST 28147-89	21
2.5. Algorithm of the Competition AES	24
2.5.1. MARS	26
2.5.2. RC6	37
2.5.3. Rijndael	41
2.5.4. TwoFish	44
2.5.5. Serpent	48
2.5.6. Results of the competition AES	51
3. System of public key encryption	52
3.1. Basic Principles	52
3.2. Practical schemes of asymmetrical encryption	56
3.2.1. Encryption algorithm RSA	56
3.2.2. RSA and Rabin Algorithms	60
3.2.3. El Gamal encryption algorithm	61
3.2.4. McEliece encryption algorithm	64
4. Methods and tools to ensure authenticity of information	67
4.1. Generalized system of digital signature	67
4.2. Electronic digital signature RSA	69
4.3. Rabin digital signature	71
4.4. El Gamal digital signature	72
4.5. DSA digital signature algorithm	76
4.6. Electronic digital signature GOST R 34.10-2001	79
5. Hash functions	80
5.1. One-way hash functions based on symmetric block algorithms	81
5.2. Self-hash algorithms	84
5.2.1. Algorithm MD5	84
5.2.2. Algorithm SHA -1	88
5.2.3. Russian standard of hash function GOST R 34.11-94	91
Conclusion	100
References	101

77-00

Учебное издание

Павел Павлович Бескид
Татьяна Михайловна Татарникова

КРИПТОГРАФИЧЕСКИЕ МЕТОДЫ ЗАЩИТЫ ИНФОРМАЦИИ
Часть 2

**Алгоритмы, методы и средства обеспечения конфиденциальности,
подлинности и целостности информации**

Учебное пособие

Редактор О.С. Крайнова
Компьютерная верстка Н.И. Афанасьевой

ЛР № 020309 от 30.12.96

Подписано в печать 13.09.10. Формат 60×90 1/16. Гарнитура Times New Roman.
Бумага офсетная. Печать офсетная. Усл. печ. л. 6,5. Тираж 250 экз. Заказ № 33/10
РГТУ, 195196, Санкт-Петербург, Малоохтинский пр., 98.
ЗАО «НПП «Система», 197045, Санкт-Петербург, Ушаковская наб., 17/1.
