Часть III

Криптографические алгоритмы

Глава 11
Математические основы

11.1 Теория информации

Современная теория информации впервые была опубликована в 1948 году Клодом Э. Шенноном (Claude Elmwood Shannon) [1431, 1432]. (Его работы были переизданы в IEEE Press [1433].) С математической точки зрения эта тема хорошо рассмотрена в [593]. В этой главе я только схем атично излагаю основные идеи.

Энтропия и неопределенность

Теория информации определяет **количество информации** в сообщении как минимальное количество бит, необходимое для кодирования всех возможных значений сообщения, считая все сообщения равновероятными. Например, для поля дня недели в базе данных достаточно использовать три бита информации, так как вся и нформация может быть закодирована 3 битами:

000 - Воскресенье
001 - Понедельник
010 - Вторник
011 - Среда
100 - Четверг
101 - Пятница
110 - Суббота
111 - Не используется

Если эта информация была бы представлена соответствующими строками ASCII символов, она заняла бы больше места в памяти, но не содержала бы больше информации. Аналогично, поле базы данных "пол" соде ржит только один бит информации, хотя эта информация может храниться как одно из двух 7-байтовых ASCII строк: "МУЖЧИНА" или "ЖЕНЩИНА".

Формально, количество информации в сообщении M измеряется **энтропией** сообщения, обозначаемое как $H(M)$. Энтропия сообщения, определяющего пол, составляет1 бит, а энтропия сообщения, определяющего день недели, немного меньше, чем 3 бита. В общем случае энтропия сообщения, измеряемая в битах, равна $\log_2 n$, где n - это количество возможных значений. При этом предполагается, что все значения равновероятны.

Энтропия сообщения также является мерой его **неопределенности**. Это количество битов открытого текста, которое нужно раскрыть в шифротексте сообщения, чтобы узнать весь открытый текст. Например, если блок шифротекста "QHP*5M " означает либо "МУЖЧИНА", либо "ЖЕНЩИНА", то неопределенность сообщения равна 1. Криптоаналитику нужно узнать только один правильно выбранный бит, чтобы раскрыть с ообщение.

Норма языка

Для данного языка **норма языка** равна

$r = H(M)/N$

где N - это длина сообщения. При больших N норма обычного английского языка принимает различные зн ачения от 1.0 бит/буква до 1.5 бит/буква. Шеннон в [1434] говорит, что энтропия зависит от длины текста. Ко нкретно он показал, что норма для 8-буквенных блоков равна 2.3 бит/буква, но ее значение падает и находится между 1.3 и 1.5 для 16-буквенных блоков. Томас Кавер (Thomas Cover) использовал игровую методику оценки и обнаружил, что энтропия равна 1.3 бит/символ [386]. (В этой книге я буду использовать значение 1.3.) **Абсолютная норма** языка равна максимальному количеству битов, которое может быть передано каждым символом при условии, что все последовательности символов равновероятны. Если в языке L символов, то абсолютная норма равна:

$R = \log_2 L$

Это максимум энтропии отдельных символов.

Для английского языка с 26 буквами абсолютная норма равна $\log_2 26$, или около 4.7 бит/буква. Вас не дол жно удивлять, что действительная норма английского языка намного меньше, чем абсолютная - естественные языки обладают высокой избыточностью. **Избыточность** языка, обозначаемая D, определяется как:

$D=R - r$

Считая, что норма английского языка равна 1.3, избыточность составит 3.4 бит/буква. Это означает, что к аждая английская буква содержит 3.4 бита избыточной информации.

У сообщения ASCII, состоящего только из английских букв, количество информации на каждый байт с о-

ставляет 1.3 бита. Значит, в каждом байте содержится 6.7 бита избыточной информации, что дает общую изб ыточность 0.84 бита информации на бит ASCII-текста и энтропию 0.16 бита информации на бит ASCII-текста. То же сообщение, набранное кодом BAUDOT, с 5 битами на символ, имеет избыточность 0.74 бита на бит и энтр опию 0.26 бита на бит. Пробелы, пунктуация, числа и форматирование изменяют эти результаты.

Безопасность криптосистемы

Шеннон определил точную математическую модель понятия безопасности криптосистемы. Смысл работы криптоаналитика состоит в определении ключа *K*, открытого текста *P* или и того, и другого. Однако, его может устроить и некоторая вероятностная информация о *P*: является ли этот открытый текст оцифрованным звуком, немецким текстом, данными электронных таблиц или еще чем-нибудь.

В реальном криптоанализе у криптоаналитика есть некоторая вероятностная информация о *P* еще до начала работы. Он, скорее всего, знает язык открытого текста. Этот язык обладает определенной, связанной с ним и з быточностью. Если это сообщения для Боба, оно, возможно, начинается словами "Дорогой Боб". Определенно, "Дорогой Боб" намного вероятнее, чем "e8T&.g [,m". Целью криптоаналитика является изменение вероятностей, связанных с каждым возможным открытым текстом. В конце концов, из груды возможных открытых текстов будет выбран один конкретный (или, по крайней мере, весьма вероятный).

Существуют криптосистемы, достигающие **совершенной безопасности**. Такой является криптосистема, в которой шифротекст не дает никакой информации об открытом тексте (кроме, возможно, его длины). Шеннон теоретически показал, что такое возможно только, если число возможных ключей также велико, как и число возможных сообщений. Другими словами, ключ должен быть не короче самого сообщения и не может испол ьзоваться повторно. Это означает, что единственной системой, которая достигает идеальной безопасности, может быть только криптосистема с одноразовым блокнотом (см. раздел 1.5).

За исключением идеально безопасных систем, шифротекст неизбежно дает определенную информацию о с ответствующем шифротексте. Хороший криптографический алгоритм сохраняет минимум этой информации, хороший криптоаналитик пользуется этой информацией для определения открытого текста.

Криптоаналитики используют естественную избыточность языка для уменьшения числа возможных откр ытых текстов. Чем избыточнее язык, тем легче его криптоанализировать. По этой причине многие криптограф ические реализации перед шифрованием используют программы сжатия для уменьшения размера текста. Сжатие уменьшает избыточность сообщения вместе с объемом работы, необходимым для его шифрования и дешифр ирования.

Энтропия криптосистемы является мерой размера пространства ключей, *K*. Она приблизительно равна лога рифму числа ключей по основанию 2:

$H(K) = \log_2 K$

Энтропия криптосистемы с 64-битовым ключом равна 64 битам, энтропия криптосистемы с 56-битовым ключом равна 56 битам. В общем случае чем больше энтропия, тем тяжелее взломать криптосистему.

Расстояние уникальности

Для сообщения длиной *n* число различных ключей, которые расшифруют шифротекст сообщения в какой-то осмысленный открытый текст на языке оригинального открытого текста (например, английском), определяется следующей формулой [712, 95]:

$2^{H(K)-nD}-1$

Шеннон [1432] определил **расстояние уникальности**, *U*, называемое также точкой уникальности, как такое приближенное количество шифротекста, для которого сумма реальной информации (энтропия) в соответству ющем открытом тексте плюс энтропия ключа шифрования равняется числу используемых битов шифротекста. Затем он показал, что имеет смысл считать, что шифротексты, которые длиннее расстояния уникальности, мо жно расшифровать только одним осмысленным способом. Шифротексты, которые заметно короче расстояния уникальности, скорее всего, можно расшифровать несколькими способами, каждый из которых может быть правилен, и таким образом обеспечить безопасность, поставив противника перед выбором правильного откр ытого текста.

Для большинства симметричных криптосистем расстояние уникальности определяется как энтропия крипт осистемы деленная на избыточность языка.

$U = H(K)/D$

Расстояние уникальности является не точным, а вероятностным значением. Оно позволяет оценить мин имальное количество шифротекста, при вскрытии которого грубой силой имеется, вероятно, только один разу мный способ дешифрирования. Обычно чем больше расстояние уникальности, тем лучше криптосистема. Для DES с 56-битовым ключом и англоязычного сообщения, записанного символами ASCII, расстояние уникальн о-

сти приблизительно равно 8.2 символа ASCII или 66 бит. В 1405-й приведены расстояния уникальности для различных длин ключа. Расстояния уникальности для некоторых классических криптосистем можно найти в [445].

Расстояние уникальности измеряет не количество криптотекста, нужного для криптоанализа, а количество криптотекста, необходимое для единственности результата криптоанализа. Криптосистема может быть вычислительно неуязвима, даже если теоретически ее возможно взломать, используя малое количество шифротекста. (Уместно вспомнить о весьма эзотерической теории релятивистской криптографии [230, 231, 232, 233, 234, 235].) Расстояние уникальности пропорционально избыточности. Если избыточность стремится к нулю, даже тривиальный шифр может не поддаться вскрытию с использованием только шифротекста.

Табл. 11-1.
Расстояния уникальности текста ASCII, зашифрованного алгоритмами с различной длиной ключа

Длина ключа (в битах)	Расстояние уникальности (в символах)
40	5.9
56	8.2
64	9.4
80	11.8
128	18.8
256	37.6

Шеннон определил криптосистему с бесконечным расстоянием уникальности, как обладающую **идеальной тайной**. Обратите внимание, что идеальная криптосистема не обязательно является совершенной, хотя совершенная криптосистема обязательно будет и идеальной. Если криптосистема обладает идеальной тайной, то даже при успешном криптоанализе останется некоторая неопределенность, является ли восстановленный открытый текст реальным открытым текстом.

Практическое использование теории информации

Хотя эти понятия имеют большое теоретическое значение, реальный криптоанализ использует их достаточно редко. Расстояние уникальности гарантирует ненадежность системы, если оно слишком мало, но его высокое значение не гарантирует безопасности. Несколько практических алгоритмов абсолютно не поддаются анализу, поведение параметров теории информации могло бы способствовать взлому некоторых шифрованных сообщений. Однако, подобные соображения теории информации иногда полезны, например, для определения в конкретном алгоритме рекомендуемого интервала изменения ключей. Криптоаналитики также используют ряд тестов не базе статистики и теории информации, чтобы выбирать наиболее перспективные направления анализа. К сожалению, большинство литературы по применению теории информации в криптоанализе остается секретной, включая основополагающую работу Алана Тьюринга (Alan Turing), написанную в 1940.

Путаница и диффузия

Двумя основными методами маскировки избыточности открытого текста сообщения, согласно Шеннону, служат путаница и диффузия [1432].

Путаница маскирует связь между открытым текстом и шифротекстом. Она затрудняет попытки найти в шифротексте избыточность и статистические закономерности. Простейшим путем создать путаницу является подстановка. В простом подстановочном шифре, например, шифре Цезаря, все одинаковые буквы открытого текста заменяются другими одинаковыми буквами шифротекста. Современные подстановочные шифры являются более сложными: длинный блок открытого текста заменяется блоком шифротекста, и способ замены меняется с каждым битом открытого текста или ключа. Такого типа подстановки обычно недостаточно - сложный алгоритм немецкой Энигмы был взломан в ходе второй мировой войны.

Диффузия рассеивает избыточность открытого текста, распространяя ее по всему шифротексту. Криптоаналитику потребуется немало времени для поиска избыточности. Простейшим способом создать диффузию является транспозиция (также называемая **перестановкой**). Простой перестановочный шифр только переставляет буквы открытого текста. Современные шифры также выполняют такую перестановку, но они также используют другие формы диффузии, которые позволяют разбросать части сообщения по всему сообщению.

Потоковые шифры используют только путаницу, хотя ряд схем с обратной связью добавляют диффузию. Блочные алгоритмы применяют и путаницу, и диффузию. Как правило, диффузию саму по себе несложно взло-

мать (хотя шифры с двойной перестановкой оказываются поустойчивее, чем другие некомпьютерные системы).

11.2 Теория сложности

Теория сложности обеспечивает методологию анализа **вычислительной сложности** различных криптографических методов и алгоритмов. Она сравнивает криптографические методы и алгоритмы и определяет их безопасность. Теория информации сообщает нам о том, что все криптографические алгоритмы (кроме однор азовых блокнотов) могут быть взломаны. Теория сложности сообщает, могут ли они быть взломаны до тепловой смерти вселенной.

Сложность алгоритмов

Сложность алгоритма определяется вычислительными мощностями, необходимыми для его выполнения. Вычислительная сложность алгоритма часто измеряется двумя параметрами: T (**временная сложность**) и S (**пространственная сложность**, или требования к памяти). И T, и S обычно представляются в виде функций от n, где n - это размер входных данных. (Существую и другие способы измерения сложности: количество случа йных бит, ширина канала связи, объем данных и т.п.)

Обычно вычислительная сложность алгоритма выражается с помощью нотации "О большого", т.е описыв ается порядком величины вычислительной сложности. Это просто член разложения функции сложности, быстрее всего растущий с ростом n, все члены низшего порядка игнорируются. Например, если временная сложность данного алгоритма равна $4n^2+7n+12$, то вычислительная сложность порядка n^2, записываемая как $O(n^2)$.

Временная сложность измеренная таким образом не зависит от реализации. Не нужно знать ни точное время выполнения различных инструкций, ни число битов, используемых для представления различных переменных, ни даже скорость процессора. Один компьютер может быть на 50 процентов быстрее другого, а у третьего шина данных может быть в два раза шире, но сложность алгоритма, оцененная по прядку величины, не изменится. Это не жульничество, при работе с алгоритмами настолько сложными, как описанные в этой книге, всем пр очим можно пренебречь (с точностью до постоянного множителя) в сравнении со сложностью по порядку вел ичины.

Эта нотация позволяет увидеть, как объем входных данных влияет на требования к времени и объему пам яти. Например, если $T= O(n)$, то удвоение входных данных удвоит и время выполнения алгоритма. Если $T=O(2^n)$, то добавление одного бита к входным данным удвоит время выполнения алгоритма.

Обычно алгоритмы классифицируются в соответствии с их временной или пространственной сложностью. Алгоритм называют **постоянным**, если его сложность не зависит от n: O(1). Алгоритм является **линейным**, если его временная сложность $O(n)$. Алгоритмы могут быть **квадратичными**, **кубическими** и т.д. Все эти алгоритмы - **полиномиальны**, их сложность - $O(n^m)$, где m - константа. Алгоритмы с полиномиальной временной сложностью называются алгоритмами **с полиномиальным временем**.

Алгоритмы, сложность которых равна $O(t^{f(n)})$, где t - константа, большая, чем 1, а $f(n)$ - некоторая полиномиальная функция от n, называются **экспоненциальными**. Подмножество экспоненциальных алгоритмов, сло жность которых равна $O(c^{f(n)})$, где где с - константа, а $f(n)$ возрастает быстрее, чем постоянная, но медленнее, чем линейная функция, называется **суперполиномиальным**.

В идеале, криптограф хотел бы утверждать, что алгоритм, лучший для взлома спроектированного алгоритма шифрования, обладает экспоненциальной временной сложностью. На практике, самые сильные утверждения, которые могут быть сделаны при текущем состоянии теории вычислительной сложности, имеют форму "все и звестные алгоритмы вскрытия данной криптосистемы обладают суперполиномиальной временной сложностью". То есть, известные нам алгоритмы вскрытия обладают суперполиномиальной временной сложностью, но пока невозможно доказать, что не может быть открыт алгоритм вскрытия с полиномиальной временной сложностью. Развитие теории вычислительной сложности возможно когда-нибудь позволит создать алгоритмы, для которых существование алгоритмов с полиномиальным временем вскрытия может быть исключено с математической точностью.

С ростом n временная сложность алгоритмов может стать настолько огромной, что это повлияет на практ ическую реализуемость алгоритма. В 9-й показано время выполнения для различных классов алгоритмов при n равном одному миллиону. В таблице игнорируются постоянные величины, но показано, почему это можно д елать.

Табл. 11-2
Время выполнения для различных классов алюритмов

Класс	Сложность	Количество операций для $n=10^6$	Время при 10^6 операций в секунду

Постоянные	O(1)	1	1 мкс
Линейные	O(n)	10^6	1 с
Квадратичные	O(n^2)	10^{12}	11.6 дня
Кубические	O(n^3)	10^{18}	32000 лет
Экспоненциальные	O(2^n)	10^{301030}	В 10^{301006} раз больше, чем время существования вселенной

При условии, что единицей времени для нашего компьютера является микросекунда, компьютер может выполнить постоянный алгоритм за микросекунду, линейный - за секунду, а квадратичный - за 11.6 дня. Выполнение кубического алгоритма потребует 32 тысяч лет, что в принципе реализуемо, компьютер, конструкция которого позволила бы ему противостоять следующему ледниковому периоду, в конце концов получил бы решение. Выполнение экспоненциального алгоритма тщетно, независимо от экстраполяции роста мощи компьютеров, параллельной обработки или контактов с инопланетным суперразумом.

Взглянем на проблему вскрытия алгоритма шифрования грубой силой. Временная сложность такого вскрытия пропорциональна количеству возможных ключей, которое экспоненциально зависит от длины ключа. Если n - длина ключа, то сложность вскрытия грубой силой равна O(2^n). В разделе 12.3 рассматривается дискуссия об использовании для DES 56-битового ключа вместо 112-битового. Сложность вскрытия грубой силой при 56-битовом ключе составляет 2^{56}, а при 112-битовом ключе - 2^{112}. В первом случае вскрытие возможно, а во втором - нет.

Сложность проблем

Теория сложности также классифицирует и сложность самих проблем, а не только сложность конкретных алгоритмов решения проблемы. (Отличным введением в эту тему являются [600, 211, 1226], см. также [1096, 27, 739].) Теория рассматривает минимальное время и объем памяти, необходимые для решения самого трудного варианта проблемы на теоретическом компьютере, известном как **машина Тьюринга**. Машина Тьюринга представляет собой конечный автомат с бесконечной лентой памяти для чтения-записи и является реалистичной моделью вычислений.

Проблемы, которые можно решить с помощью алгоритмов с полиномиальным временем, называются решаемыми, потому что для разумных входных данных обычно могут быть решены за разумное время. (Точное определение "разумности" зависит от конкретных обстоятельств.) Проблемы, которые невозможно решить за полиномиальное время, называются нерешаемыми, потому что вычисление их решений быстро становится невозможным. Нерешаемые проблемы иногда называют **трудными**. Проблемы, которые могут быть решены только с помощью суперполиномиальных алгоритмов, вычислительно нерешаемы, даже при относительно малых значениях n.

Что еще хуже, Алан Тьюринг доказал, что некоторые проблемы **принципиально неразрешимы**. Даже отвлекаясь от временной сложности алгоритма, невозможно создать алгоритм решения этих проблем.

Проблемы можно разбить на классы в соответствии со сложностью их решения. Самые важные классы и их предполагаемые соотношения показаны на 10-й. (К несчастью, лишь малая часть этих утверждений может быть доказана математически.)

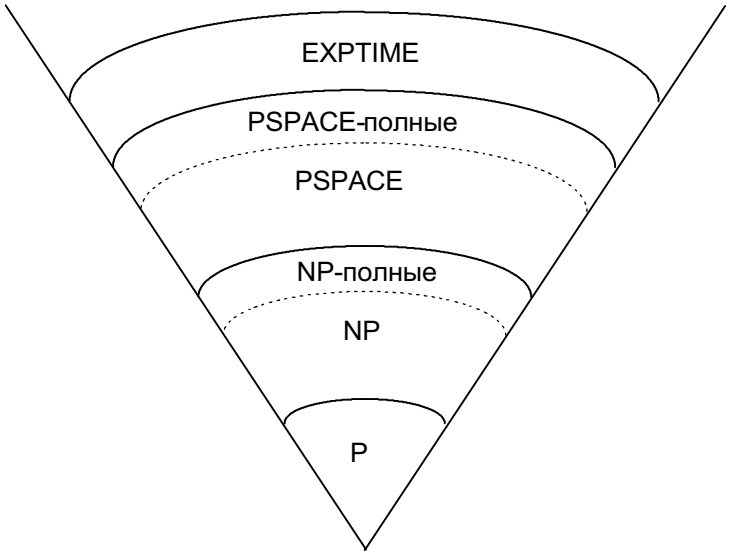


Рис. 11-1. Классы сложности

Находящийся в самом низу класс **P** состоит из всех проблем, которые можно решить за полиномиальное время. Класс **NP** - из всех проблем, которые можно решить за полиномиальное время только на недетермин ированной машине Тьюринга: вариант обычной машины Тьюринга, которая может делать предположения. М ашина предполагает решение проблемы - либо "удачно угадывая", либо перебирая все предположения пара ллельно - и проверяет свое предположение за полиномиальное время.

Важность **NP** в криптографии состоит в следующем: многие симметричные алгоритмы и алгоритмы с о ткрытыми ключами могут быть взломаны за недетерминированное полиномиальное время. Для данного шифр отекста *C*, криптоаналитик просто угадывает открытый текст, *X*, и ключ, *k*, и за полиномиальное время выполн яет алгоритм шифрования со входами *X* и *k* и проверяет, равен ли результат *C*. Это имеет важное теоретическое значение, потому что устанавливает верхнюю границу сложности криптоанализа этих алгоритмов. На практике, конечно же, это выполняемый за полиномиальное время детерминированный алгоритм, который и ищет кри птоаналитик. Более того, этот аргумент неприменим ко всем классам шифров, конкретно, он не применим для одноразовых блокнотов - для любого *C* существует множество пар *X*, *k*, дающих *C* при выполнении алгоритма шифрования, но большинство этих *X* представляют собой бессмысленные, недопустимые открытые тексты.

Класс **NP** включает класс **P**, так как любая проблема, решаемая за полиномиальное время на детерминир ованной машине Тьюринга, будет также решена за полиномиальное время на недетерминированной машине Тьюринга, просто пропускается этап предпол ожения.

Если все **NP** проблемы решаются за полиномиальное время на детерминированной машине, то **P** = **NP**. Хотя кажется очевидным, что некоторые **NP** проблемы намного сложнее других (вскрытие алгоритма шифрования грубой силой против шифрования произвольного блока шифротекста), никогда не было доказано, что **P** ≠ **NP** (или что **P** = **NP**). Однако, большинство людей, работающих над теорией сложности, убеждены, что эти классы неравны.

Что удивительно, можно доказать, что конкретные **NP**-проблемы настолько же трудны, как и любая пробл ема этого класса. Стивен Кук (Steven Cook) доказал [365], что проблема Выполнимости (Satisfiability problem, дано правильное логическое выражение, существует ли способ присвоить правильные значения входящим в него переменным так, чтобы все выражение стало истиной?) является **NP-полной**. Это означает, что, если про блема Выполнимости решается за полиномиальное время, то **P** = **NP**. Наоборот, если может быть доказано, что для любой проблемы класса **NP** не существует детерминированного алгоритма с полиномиальным временем решения, доказательство покажет, что и для проблемы Выполнимости не существует детерминированного алг оритма с полиномиальным временем решения. В **NP** нет проблемы труднее, чем проблема Выполнимости.

С тех пор, как основополагающая работа Кука была опубликована, было показано, что существует множес тво проблем, эквивалентных проблеме Выполнимости, сотни их перечислены в [600], ряд примеров приведен ниже. Из-за эквивалентности я полагаю, что эти проблемы также являются **NP-полными**, они входят в класс NP и так же сложны, как и любая проблема класса **NP**. Если бы была доказана их решаемость за детерминир ованное полиномиальное время, вопрос **P** против **NP** был бы решен. Вопрос, верно ли **P** = **NP**, является цен тральным нерешенным вопросом теории вычислительной сложности, и не ожидается, что он будет решен в ближайшее время. Если кто-то покажет, что **P** = **NP**, то большая часть этой книги станет ненужной: как объя снялось ранее многие классы шифров тривиально взламываются за недетерминированное полиномиальное вр е-

мя. Если **P** = **NP**, то они вскрываются слабыми, детерминированными алгоритмами.

Следующим в иерархии сложности идет класс **PSPACE**. Проблемы класса **PSPACE** могут быть решены в полиномиальном пространстве, но не обязательно за полиномиальное время. **PSPACE** включает **NP**, но ряд проблем **PSPACE** кажутся сложнее, чем **NP**. Конечно, и это пока недоказуемо. Существует класс проблем, так называемых **PSPACE-полных**, обладающих следующим свойством: если любая из них является **NP**-проблемой, то **PSPACE** = **NP**, и если любая из них является **P**-проблемой, то **PSPACE** = **P**.

И наконец, существует класс проблем **EXPTIME**. Эти проблемы решаются за экспоненциальное время. Может быть действительно доказано, что **EXPTIME-полные** проблемы не могут быть решены за детерминированное полиномиальное время. Также показано, что **P** не равно **EXPTIME**.

NP-полные проблемы

Майкл Кэри (Michael Carey) и Дэвид Джонсон (David Johnson) составили список более чем 300 NP-полных проблем [600]. Вот некоторые:

— Проблема путешествующего коммивояжера. Путешествующему коммивояжеру нужно посетить различные города, используя только один бак с горючим (существует максимальное расстояние, которое он может проехать). Существует ли маршрут, позволяющий ему посетить каждый голод только один раз, и с-пользуя этот единственный бак с горючим? (Это обобщение проблемы гамильтонова пути - см. раздел 5.1.)

— Проблема тройного брака. В комнате n мужчин, n женщин и n чиновников (священников, раввинов, кого угодно). Есть список разрешенных браков, записи которого состоят из одного мужчины, одной женщины и одного регистрирующего чиновника. Дан этот список троек, возможно ли построить n браков так, чтобы любой либо сочетался браком только с одним человеком или регистрировал только один брак?

— Тройная выполнимость. Есть список n логических выражений, каждое с тремя переменными. Например: если (x и y) то z, (x и w) или (не z), если ((не u и не x) или (z и (u или не x))) то (не z и u) или x), и т.д. Существует ли правильные значения всех переменных, чтобы все утверждения были истинными? (Это частный случай упомянутой выше проблемы Выполнимости.)

11.3 Теория чисел

Это не книга по теории чисел, поэтому я только набросаю ряд идей, используемых в криптографии. Если вам нужно подробное математическое изложение теории чисел, обратитесь к одной из этих книг: [1430, 72, 1171, 12, 959, 681, 742, 420]. Моими любимыми книгами по математике конечных полей являются [971, 1042]. См. также [88, 1157, 1158, 1060].

Арифметика вычетов

Вы все учили математику вычетов в школе. Иногда ее называли "арифметикой часов". Если Милдред сказала, что она будет дома к 10:00, и опоздала на 13 часов, то когда она придет домой, и на сколько лет отец лишит ее водительских прав? Это арифметика по модулю 12. Двадцать три по модулю 12 равно 11.

(10 + 13) mod 12 = 23 mod 12 = 11 mod 12

Другим способом записать это является утверждение об эквивалентности 23 и 11 по модулю 12:

10 + 13 ≡ 11 (mod 12)

В основном, $a \equiv b \pmod{n}$, если $a = b + kn$ для некоторого целого k. Если a неотрицательно и b находится между 0 и n, можно рассматривать b как остаток при делении a на n. Иногда, b называется **вычетом** a по модулю n. Иногда a называется **конгруэнтным** b по модулю n (знак тройного равенства, ≡, обозначает конгруэнтность). Одно и то же можно сказать разными способами.

Множество чисел от 0 до n-1 образует то, что называется **полным множеством вычетов** по модулю n. Это означает, что для любого целого a, его остаток по модулю n является некоторым числом от 0 до n-1.

Операция a mod n обозначает остаток от a, являющийся некоторым целым числом от 0 до n-1. Эта операция называется **приведением по модулю**. Например, 5 mod 3 = 2.

Это определение mod может отличаться от принятого в некоторых языках программирования. Например, оператор получения остатка в языке PASCAL иногда возвращает отрицательное число. Он возвращает число между -(n-1) и n-1. В языке C оператор % возвращает остаток от деления первого выражения на второе, оно может быть отрицательным числом, если любой из операндов отрицателен. Для всех алгоритмов в этой книге проверяйте, что вы добавляете n к результату операции получения остатка, если она возвращает отрицательное число.

Арифметика остатков очень похожа на обычную арифметику: она коммутативна, ассоциативна и дистриб утивна. Кроме того, приведение каждого промежуточного результата по модулю n дает тот же результат, как и выполнение всего вычисления с последующим приведением конечного результата по модулю n.

$(a + b) \bmod n == ((a \bmod n) + (b \bmod n)) \bmod n$

$(a - b) \bmod n == ((a \bmod n) - (b \bmod n)) \bmod n$

$(a * b) \bmod n == ((a \bmod n) * (b \bmod n)) \bmod n$

$(a * (b+c)) \bmod n == (((a*b) \bmod n) + ((a*c) \bmod n)) \bmod n$

Вычисление mod n часто используется в криптографии, так как вычисление дискретных логарифмов и ква дратных корней mod n может быть нелегкой проблемой. Арифметика вычетов, к тому же, легче реализуется на компьютерах, поскольку она ограничивает диапазон промежуточных значений и результата. Для k-битовых вычетов n, промежуточные результаты любого сложения, вычитание или умножения будут не длиннее, чем 2 k бит. Поэтому в арифметике вычетов мы можем выполнить возведение в степень без огромных промежуточных р езультатов. Вычисление степени некоторого числа по м одулю другого числа,

$a^x \bmod n$,

представляет собой просто последовательность умножений и делений, но существуют приемы, ускоряющие это действие. Один из таких приемов стремится минимизировать количество умножений по модулю, другой - оптимизировать отдельные умножения по модулю. Так как операции дистрибутивны, быстрее выполнить возв едение в степень как поток последовательных умножений, каждый раз получая вычеты. Сейчас вы не чувствуете разницы, но она будет заметна при умножении 200-битовых чисел.

Например, если вы хотите вычислить $a^8 \bmod n$, не выполняйте наивно семь умножений и одно приведение по модулю:

$(a * a * a * a * a * a * a * a) \bmod n$

Вместо этого выполните три меньших умножения и три меньших приведения по м одулю:

$((a^2 \bmod n)^2 \bmod n)^2 \bmod n$

Точно также,

$a^{16} \bmod n = (((a^2 \bmod n)^2 \bmod n)^2 \bmod n)^2 \bmod n$

Вычисление a^x, где x не является степенью 2, ненамного труднее. Двоичная запись представляет x в виде суммы степеней 2: 25 - это бинарное 11001, поэтому 25 = 24 + 23 + 20. Поэтому

$a^{25} \bmod n = (a*a^{24}) \bmod n = (a* a^8*a^{16}) \bmod n =$

$= (a*((a^2)^2)^2*(((a^2)^2)^2)^2) \bmod n = (a*(((a*a^2)^2)^2)^2) \bmod n$

С продуманным сохранением промежуточных результатов вам понадобится только шесть умножений:

$(((((((a^2 \bmod n)* a)^2 \bmod n)^2 \bmod n)^2 \bmod n)^2 \bmod n)^2 *a) \bmod n$

Такой прием называется **цепочкой сложений** [863], или методом двоичных квадратов и умножения. Он и спользует простую и очевидную цепочку сложений, в основе которой лежит двоичное представление числа. На языке С это выглядит следующим образом:

```
unsigned long qe2(unsigned long x, unsigned long y, unsigned long n) {
     unsigned long s, t, u;
     int i;
     s=1; t=x; u=y;
     while (u) {
             if(u&1) s=(s*t)%n;
             u>>1;
             t=(t*t)%n;
     }
     return(s)
}
```

А вот другой, рекурсивный, алгоритм:

```
unsigned long fast_exp(unsigned long x, unsigned long y, unsigned long N) {
     unsigned long tmp;
```

```
    if(y==1) return(x % N);
    if (1^(x&1)) {
            tmp= fast_exp(x,y/2,N);
            return ((tmp*tmp)%N);
    else {
            tmp = fast_exp(x,(y-1)/2,N);
            tmp = (tmp*tmp)%N;
            tmp = (tmp*x)%N;
            return (tmp);
    }
}
```

Этот метод уменьшает количество операций, в среднем, до 1.5* k операций, где k - длина числа x в битах. Найти способ вычисления с наименьшим количеством операций - трудная проблема (было доказано, что посл е-довательность должна содержать не меньше k-1 операций), но нетрудно снизить число операций до 1.1* k или даже лучше при больших k.

Эффективным способом много раз выполнять приведение по модулю для одного n является **метод Монтгомери** [1111]. Другой метод называется **алгоритмом Баррета** [87]. Эффективность описанного алгоритма и этих двух методов рассматривается в [210]: алгоритм, рассмотренный мною, является наилучшим для едини ч-ного приведения по модулю, алгоритм Баррета - наилучшим для малых аргументов, а метод Монтгомери - на и-лучшим для обычного возведения в степень по модулю. (Метод Монтгомери также использует преимущество малых показателей степени, используя прием, называющийся смешанной арифметикой.)

Операция, обратная возведению в степень по модулю n, вычисляет **дискретный логарифм**. Я дальше вкратце рассмотрю эту операцию.

Простые числа

Простым называется целое число, большее единицы, единственными множителями которого является 1 и оно само: оно не делится ни на одно другое число. Два - это простое число. Простыми являются и 73, 2521, 2365347734339 и 2^{756839}-1. Существует бесконечно много простых чисел. Криптография, особенно криптография с открытыми ключами, часто использует большие простые числа (512 бит и даже бол ьше).

Евангелос Кранакис (Evangelos Kranakis) написал отличную книгу по теории чисел, простым числам и их применению в криптографии [896]. Паула Рибенбойм (Paula Ribenboim) написала две отличных справочных работы по простым числам вообще [1307, 1308].

Наибольший общий делитель

Два числа называются **взаимно простыми**, если у них нет общих множителей кроме 1. Иными словами, е с-ли **наибольший общий делитель** a и n равен 1. Это записывается как:

НОД(a,n)=1

Взаимно просты числа 15 и 28. 15 и 27 не являются взаимно простыми, а 13 и 500 - являются. Простое число взаимно просто со всеми другими числами, кроме ч исел, кратных данному простому числу.

Одним из способов вычислить наибольший общий делитель двух чисел является **алгоритм Эвклида**. Эвк-лид описал этот алгоритм в своей книге, *Элементы*, написанной в 300 году до нашей эры. Он не изобрел его. Историки считают, что этот алгоритм лет на 200 старше. Это самый древний нетривиальный алгоритм, который дошел до наших дней, и он все еще хорош. Кнут описал алгоритм и его современные модификации в [863]. На языке C:

```
/* возвращает НОД (gcd) x и y */
int gcd (int x, int y) {
    int g;
    if (x < 0)
            x = -x;
    if (y < 0)
            y = -y;
    if (x + y == 0 )
            ERROR ;
```

```
    g = y;
    while (x > 0) {
            g = x;
            x = y % x;
            y = g;
    }
    return g;
}
```

Этот алгоритм можно обобщить для получения НОД массива *m* чисел:

```
/* возвращает НОД (gcd) xl, x2...xm */
int multiple_gcd (int m, int *x) {
    slze_t i;
    int g;
    if (m < 1)
            return 0;
    g = x [0];
    for (i=l; i<m; ++i) {
            g = gcd(g, x[i]);
/* оптимизация, так как для случайных x[i], g==l в 60% случаев: */
            if (g == 1)
            return 1;
    }
    return g;
}
```

Обратные значения по модулю

Помните, что такое обратные значения? Обратное значение для 4 - 1/4, потому что 4*1/4 =1. В мире вычетов проблема усложняется:

$4*x = 1 \pmod 7$

Это уравнение эквивалентно обнаружению x и k, таких что

$4x = 7k + 1$

где x и k - целые числа. Общая задача состоит в нахождении x, такого что

$1 = (a*x) \bmod n$

Это также можно записать как

$a^{-1} \equiv x \pmod n$

Проблему обратных значений по модулю решить нелегко. Иногда у нее есть решение, иногда нет. Например, обратное значение 5 по модулю 14 равно 3. С другой стороны у числа 2 нет обратного значения по модулю 14.

В общем случае у уравнения $a^{-1} \equiv x \pmod n$ существует единственное решение, если a и n взаимно просты. Если a и n не являются взаимно простыми, то $a^{-1} \equiv x \pmod n$ не имеет решений. Если n является простым числом, то любое число от 1 до n -1 взаимно просто с n и имеет в точности одно обратное значение по модулю n.

Так, хорошо. А теперь как вы собираетесь искать обратное значение a по модулю n? Существует два пути. Обратное значение a по модулю n можно вычислить с помощью алгоритма Эвклида. Иногда это называется расширенным алгоритмом Эвклида.

Вот этот алгоритм на языке C++:

```
#define isEven(x) ((x & 0x01) == 0)
#define isOdd(x)  (x& 0x01)
#define swap(x,y) (x^= y, y^= x, x^= y)
void ExtBinEuclid(int *u, int *v, int *u1, int *u2, int *u3) {
    // предупреждение: u и v будут переставлены, если u<v
int k, tl, t2, t3;
```

```
if (*u < *v) swap(*u<,*v);
for (k = 0; isEven(*u) && isEven(*v); ++k) {
*u>>=1; *v >>1;
}
*u1 = 1; *u2 = 0; *u3 = *u; t1 = *v; t2 = *u - 1; t3 = *v;
do {
do {
if (isEven(*u3)) {
if (isOdd(*ul) || isOdd(*u2)) {
*u1 += *v; *u2 += *u;
                            }
*ul >>* 1; *u2 >>= 1; *u3 >>= 1;
                    }
if (isEven(t3) || *u3 < t3) {
swap(*ul,tl); smap(*u2,t2); smap(*u3,t3);
                    }
} while (isEven(*u3));
while (*ul < tl || *u2 < t2) {
*ul += *v; *u2 += *u;
            }
            ul -= tl; *u2 -= t2; *u3 -= t3;
} while (t3 > 0);
while (*ul >= *v && *u2 >= *u) {
*ul>l -= *v; *u2 -= *u;
     }
*u <<= k; *v <<= k; *u3 << k;
}
main(int argc, char **argv) {
int a, b, gcd;
if (argc < 3) {
cerr << "как использовать: xeuclid u v" << end1;
return -1;
     }
     int u = atoi(argv[1]);
int v = atoi(argv[2]);
if (u <= 0 || v <= 0 ) {
     cerr << "Аргумент должен быть положителен!" << end1;
return -2;
     }
// предупреждение: u и v будут переставлены если u < v
ExtBinEuclid(&u, &v, &a, &b, &gcd);
cout << a <<" * " << u << " + (-"
<< b << ") * " << v << " = " << gcd << end1;
if (gcd == 1)
cout << "Обратное значение " << v << " mod " << u << " is: "
<< u - b << end1;
return 0;
}
```

Я не собираюсь доказывать, что это работает, или приводить теоретическое обоснование. Подробности можно найти в [863] или в любой из приведенных ранее работ по теории чисел.

Алгоритм итеративен и для больших чисел может работать медленно. Кнут показал, что среднее число в ы-

полняемых алгоритмом делений равно:

$0.843*\log_2(n) + 1.47$

Решение для коэффициентов

Алгоритм Эвклида можно использовать и для решения следующих проблем: дан массив из m переменных $x_1, x_2, ..., x_m$, найти массив m коэффициентов, $u_1, u_2, ..., u_m$, таких что

$u_1 * x_1 + ... + u_m * x_m, = 1$

Малая теорема Ферма

Если m - простое число, и a не кратно m, то **малая теорема Ферма** утверждает

$a^{m-1} \equiv 1 \pmod m$

(Пьер де Ферма (Pierre de Fermat), французский математик, жил с 1601 по 1665 год. Эта теорема не имеет ничего общего с его знаменитой теоремой.)

Функция Эйлера

Существует другой способ вычислить обратное значение по модулю n, но его не всегда возможно использовать. **Приведенным множеством остатков** mod n называется подмножество полного множества остатков, члены которого взаимно просты с n. Например, приведенное множество остатков mod 12 - это {1, 5, 7, 11}. Если n - простое число, то приведенное множество остатков mod n - это множество всех чисел от 1 до n-1. Для любого n, не равного 1,число 0 никогда не входит в приведенное множество остатков.

Функция Эйлера, которую также называют функцией фи Эйлера и записывают как $\phi(n)$, - это количество элементов в приведенном множестве остатков по модулю n. Иными словами, $\phi(n)$ - это количество положительных целых чисел, меньших n и взаимно простых с n (для любого n, большего 1). (Леонард Эйлер (Leonhard Euler), швейцарский математик, жил с 1707 по 1783 год.)

Если n - простое число, то $\phi(n) = n$-1. Если $n = pq$, где p и q -простые числа, то $\phi(n) = (p - 1)(q - 1)$. Эти числа появляются в некоторых алгоритмах с открытыми ключами, и вот почему. В соответствии с обобщением Эйлера малой теоремы Ферма, если НОД(a,n) = 1, то

$a^{\phi(n)} \bmod n = 1$

Теперь легко вычислить $a^{-1} \bmod n$:

$x = a^{\phi(n)-1} \bmod n$

Например, какое число является обратным для 5 по модулю 7? Так как 7 - простое число, $\phi(7) = 7 - 1 = 6$. Итак, число, обратное к 5 по модулю 7, равно

$5^{6-1} \bmod 7 = 5^5 \bmod 7 = 3$

Эти методы вычисления обратных значений можно расширить для более общей проблемы нахождения x (если НОД(a,n) = 1):

$(a*x) \bmod n = b$

Используя обобщение Эйлера, решаем

$x = (b* a^{\phi(n)-1}) \bmod n$

Используя алгоритм Эвклида, находим

$x = (b* (a^{-1} \bmod n)) \bmod n$

В общем случае для вычисления обратных значений алгоритм Эвклида быстрее, чем обобщение Эйлера, особенно для чисел длиной порядка 500 бит. Если НОД(a,n) $\neq$ 1, не все потеряно. В этом общем случае ($a*x$) mod n=b, может иметь или несколько решений, или ни одного.

Китайская теорема об остатках

Если известно разложение числа n на простые сомножители, то для решения полной системы уравнений можно воспользоваться Китайской теоремой об остатках. Основной вариант этой теоремы был открыт в первом веке китайским математиком Сун Цзе.

В общем случае, если разложение числа n на простые сомножители представляет собой $p_1*p_2*...*p_t$, то система уравнений

($x \bmod p_i$) = a_i, где i = 1, 2, . . . , t

имеет единственное решение, x, меньшее n. (Обратите внимание, что некоторые простые числа могут появляться несколько раз. Например, p_1 может быть равно p_2.) Другими словами, число (меньшее, чем произведение нескольких простых чисел) однозначно определяется своими остатками от деления на эти простые числа.

Например, возьмем простые числа 3 и 5, и 14 в качестве заданного числа. 14 mod 3 = 2, и 14 mod 5 = 4. С уществует единственное число, меньшее 3*5 = 15, с такими остатками: 14. Два остатка однозначно определяют число.

Поэтому для произвольного $a < p$ и $b < q$ (где p и q - простые числа), существует единственное число x, меньшее pq, такое что

$x \equiv a \pmod p$, и $x \equiv b \pmod q$

Для получения x сначала воспользуемся алгоритмом Эвклида, чтобы найти u, такое что

$u*q \equiv 1 \pmod p$

Затем вычислим:

$x = (((a - b) * u) \bmod p) * q + b$

Вот как выглядит Китайская теорема об остатках на языке С:

```
/* r - это количество элементов в массивах m and u;
m - это массив (попарно взаимно простых) модулей
u - это массив коэффициентов
возвращает значение n, такое что n == u[k]%m[k] (k=0..r-1) и
n < [m[0]*m[l]*...*m[r-1]
*/
/* Получение функции Эйлера (totient) остается упражнением для читателя. */
int Chinese_remainder (size_t r, int *m, int *u) {
size_t i;
int modulus;
int n;
modulus=1;
for (i=0; i<r; ++i)
modulus*=m[i];
n=0;
for (i=0; i<r; ++i) {
n+=u[i] * modexp(modulus/m[i]*totient(m[i]),m[i]);
n %= modulus;
     }
return n;
}
```

Обращение Китайской теоремы об остатках может быть использовано для решения следующей проблемы: если p и q - простые числа, и p меньше q, то существует единственное x, меньшее, чем pq, такое что

$a \equiv x \pmod p$, и b ≡ x (mod q)

Если $a \geq b \bmod p$, то

$x = (((a - (b \bmod p)) * u) \bmod p) * q + b$

Если $a < b \bmod p$, то

$x = (((a + p - (b \bmod p))*u) \bmod p)*q + b$

Квадратичные вычеты

Если p - простое число, и a больше 0, но меньше p, то a представляет собой квадратичный вычет по модулю p, если

$x^2 \equiv a \pmod p$, для некоторых x

Не все значения *a* соответствуют этому требованию. Чтобы *a* было квадратичным вычетом по *n*, оно должно быть квадратичным вычетом по модулю всех простых сомножителей *n*. Например, если *p* = 7, квадратичными вычетами являются числа 1, 2, и 4:

$1^2 = 1 \equiv 1 \pmod 7$

$2^2 = 4 \equiv 4 \pmod 7$

$3^2 = 9 \equiv 2 \pmod 7$

$4^2 = 16 \equiv 2 \pmod 7$

$5^2 = 25 \equiv 4 \pmod 7$

$6^2 = 36 \equiv 1 \pmod 7$

Заметьте, что каждый квадратичный вычет дважды появляется в этом списке. Значений *x*, удовлетворяющих любому из следующих уравнений, не существует:

$x^2 \equiv 3 \pmod 7$

$x^2 \equiv 5 \pmod 7$

$x^2 \equiv 6 \pmod 7$

Эти числа - 3, 5 и 6 - не являются квадратичными вычетами по модулю 7.

Хотя я этого и не делаю, несложно доказать, что когда *p* нечетно, существует в точности (*p* - 1)/2 квадратичных вычетов по модулю *p*, и столько же чисел, не являющихся квадратичными вычетами по модулю *p*. Кроме того, если *a* - это квадратичный вычет по модулю *p*, то у *a* в точности два квадратных корня, один между 0 и (*p*-1)/2, а второй - между (*p* - 1)/2 и (*p* - 1). Один из этих квадратных корней одновременно является квадрати чным остатком по модулю *p*, он называется **главным квадратным корнем**.

Если *n* является произведением двух простых чисел, *p* и *q*, то существует ровно (*p* - l)(*q* - 1)/4 квадратичных вычетов по модулю *n*. Квадратичный вычет по модулю *n* является совершенным квадратом по модулю *n*, потому что для того, чтобы быть квадратом по модулю *n*, вычет должен быть квадратом по модулю *p* и квадратом по модулю *q*. Например, существует одиннадцать квадратичных остатков mod 35: 1, 4, 9, 11, 15, 16, 21, 25, 29 и 30. У каждого квадратичного вычета ровно четыре квадратных корня.

Символ Лежандра

Символ Лежандра, L(*a*,*p*), определен, если *a* - это любое целое число, а *p* - простое число, большее, чем 2. Он равен 0, 1 или -1.

L(*a*,*p*) = 0, если а делится на p.

L(*a*,*p*) = 1, если а - квадратичный вычет по модулю *p*.

L(*a*,*p*) = -1, если а не является квадратичным вычетом по модулю *p*.

L(*a*,*p*) можно рассчитать следующим образом:

$L(a,p) = a^{(p-1)/2} \bmod p$

Или можно воспользоваться следующим алгоритмом:

1. Если а = 1, то L(*a*,*p*) = 1
2. Если а четно, то $L(a,p) = L(a/2,p) * (-1)^{(з^2-1)/8}$
3. Если а нечетно (и ≠ 1), то $L(a,p) = L(p \bmod a, p) * (-1)^{(a-1)(p-1)/4}$

Обратите внимание, что этот метод также является эффективным способом определить, является ли *a* квадратичным вычетом по модулю *p* (для простого числа *p*).

Символ Якоби

Символ Якоби, J(*a*,*n*), представляет собой обобщение символа Лежандра на составные модули, он определ яется для любого целого *a* и любого нечетного целого *n*. Функция удобна при проверке на простоту. Символ Як оби является функцией на множестве полученных вычетов делителей *n* и может быть вычислен по различным формулам [1412]. Вот один из сп особов:

Определение 1: J(*a*,*n*) определен, только если *n* нечетно.

Определение 2: J(0,*n*) = 0.

Определение 3: Если n - простое число, то символ Якоби J(a,n) = 0, если a делится на n.

Определение 4: Если n - простое число, то символ Якоби J(a,n) = 1, если a - квадратичный вычет по модулю n.

Определение 5: Если n - простое число, то символ Якоби J(a,n) = -1, если a не является квадратичным вычетом по модулю n.

Определение 6: Если n - составное число, то символ Якоби J(a,n) = J(a,p_1)* ... * J(a,p_m), где p_1, ... , p_m - это разложение n на простые сомножители.

Следующий алгоритм рекурсивно рассчитывает символ Якоби:

Правило 1: J(1,n) = 1

Правило 2: J(a*b,n) = J(a,n)* J(b,n)

Правило 3: J(2,n) =, если (n^2-1) /8 нечетно, и -1 в противном случае

Правило 4: J(a,n)= J((a mod n),n)

Правило 5: J(a, b_1*b_2) = J(a, b_1)* J(a, b_2)

Правило 6: Если наибольший общий делитель a и b = 1, а также a и b нечетны:

Правило 6a: J(a,b)= J(b, a), если (a - l)(b - 1)/4 четно

Правило 6b: J(a,b)= -J(b, a), если (a - l)(b - 1)/4 нечетно

Вот алгоритм на языке C:

```
/* Этот алгоритм рекурсивно вычисляет символ Якоби */
int jacobi(int a, int b) {
int g;
     assert(odd(b));
     if (a >= b) a %= b; /* по правилу 4 */
     if (a == 0) return 0; /* по определению 1 */
     if (a == 1) return 1; /* по правилу 1 */
     if (a < 0)
            if ((b-l)/2 % 2 == 0)
                   return jacobi(-a,b);
            else
                   return -jacobi(-a,b);
     if (a % 2 == 0) /* a четно */
            if (((b*b -1)/8) % 2 == 0)
                   return +jacobi(a/2,b);
            else
                   return -jacobi(a/2,b); /* по правилам 3 и 2 */
     g = gcd(a,b);
     assert(odd(a)); /* это обеспечивается проверкой (a % 2 == 0) */
     if (g == a) /* b делится на a */
            return 0; /* по правилу 5 */
     else if (g != 1)
            return jacobi(g,b)*jacobi(a/g,b); /* по правилу 2 */
     else if (((a-l)*(b-l)/4) % 2 == 0)
            return +jacobi(b,a); /* по правилу 6a */
     else
            return -jacobi(b,a); /* по правилу 6b */
}
```

Если заранее известно, что n - простое число, вместо использования предыдущего алгоритма просто вычислите a((n-1)/2) mod n, в этом случае J(a,n) эквивалентен символу Лежандра.

Символ Якоби нельзя использовать для определения того, является ли a квадратичным вычетом по модулю

n (если, конечно, n не является простым числом). Обратите внимание, что если J(a,n) = 1 и n - составное число, то утверждение, что a является квадратичным вычетом по модулю n, не обязательно будет истиной. Например:

J(7,143) = J(7,11)* J(7,13) = (-1)(-1) = 1

Однако не существует таких целых чисел x, что $x^2 \equiv 7 \pmod{143}$.

Целые числа Блюма

Если p и q - два простых числа, конгруэнтных 3 по модулю 4, то $n = pq$ иногда называют **целым числом Блюма**. Если n - это целое число Блюма, у каждого квадратичного вычета ровно четыре квадратных корня, один из которых также является квадратом - это главный квадратный корень. Например, главный квадратный корень 139 mod 437 - это 24. Остальные три корня - это 185, 252 и 413.

Генераторы

Если p - простое число, и g меньше, чем p, то g называется **генератором** по модулю p, если для каждого числа b от 1 до p - 1 существует некоторое число a, что $g^a \equiv b \pmod{p}$.

Иными словами, g является **примитивом** по отношению к p. Например, если p = 11, то 2 - это генератор по модулю 11:

$2^{10} = 1024 \equiv 1 \pmod{11}$

$2^1 = 2 \equiv 2 \pmod{11}$

$2^8 = 256 \equiv 3 \pmod{11}$

$2^2 = 4 \equiv 4 \pmod{11}$

$2^4 = 16 \equiv 5 \pmod{11}$

$2^9 = 512 \equiv 6 \pmod{11}$

$2^7 = 128 \equiv 7 \pmod{11}$

$2^3 = 8 \equiv 8 \pmod{11}$

$2^6 = 64 \equiv 9 \pmod{11}$

$2^5 = 32 \equiv 10 \pmod{11}$

Каждое число от 1 до 10 может быть представлено как $2^a \pmod{p}$. Для p = 11 генераторами являются 2, 6, 7 и 8. Другие числа не являются генераторами. Например, генератором не является число 3, потому что не сущ е-ствует решения для

$3^a \equiv 2 \pmod{11}$

В общем случае проверить, является ли данное число генератором, нелегко. Однако задщача упрощается, е с-ли известно разложение на множители для p - 1. Пусть q_1, q_2, ... , q_n - это различные простые множители p - 1. Чтобы проверить, является ли число g генератором по модулю p, вычислите

$g^{(p-1)/q} \bmod p$

для всех значений $q = q_1, q_2, \ldots, q_n$.

Если это число равно 1 для некоторого q, то g не является генератором. Если для всех значений q рассчитан-ное значение не равно 1, то g - это генератор.

Например, пусть p = 11. Простые множители p - 1 = 10 - это 2 и 5. Для проверки того, является ли число 2 генератором, вычислим:

$2^{(11-1)/5} \pmod{11} = 4$

$2^{(11-1)/2} \pmod{11} = 10$

Ни один из ответов не равен 1, поэтому 2 - это генератор.

Проверим, является генератором ли число 3:

$3^{(11-1)/5} \pmod{11} = 9$

$3^{(11-1)/2} \pmod{11} = 1$

Следовательно, 3 - это не генератор.

При необходимости обнаружить генератор по модулю p просто случайно выбирайте число от 1 до p - 1 и проверяйте, не является ли оно генератором. Генераторов достаточно, поэтому один из них вы, скорее всего, найдете быстро.

Вычисление в поле Галуа

Не тревожьтесь, все это мы уже делали. Если n - простое число или степень большого простого числа, то мы получаем то, что математики называют **конечным полем**. В честь этого мы используем p вместо n. В действительности этот тип конечного поля настолько замечателен, что математики дали ему собственное имя - **поле Галуа**, обозначаемое как GF(p). (В честь Эвариста Галуа, французского математика, жившего в девятнадцатом веке и успевшего значительно продвинуть теорию чисел, прежде чем в 20 лет он был убит на дуэли.)

В поле Галуа определены сложение, вычитание, умножение и деление на ненулевые элементы. Существует нейтральный элемент для сложения - 0 - и для умножения - 1. Для каждого ненулевого числа существует единственное обратное число (это не было бы так, если бы p не было бы простым числом). Выполняются коммутативный, ассоциативный и дистрибутивный законы.

Арифметика поля Галуа широко используется в криптографии. В нем работает вся теория чисел, поле содержит числа только конечного размера, при делении отсутствуют ошибки округления. Многие криптосистемы основаны на GF(p), где p - это большое простое число.

Чтобы еще более усложнить вопрос, криптографы также используют арифметику по модулю **неприводимых** многочленов степени n, коэффициентами которых являются целые числа по модулю q, где q - это простое число. Эти поля называются GF(qn). Используется арифметика по модулю $p(x)$, где $p(x)$ - это неприводимый многочлен степени n.

Математическая теория, стоящая за этим, выходит далеко за рамки этой книги, хотя я и опишу ряд криптосистем, использующих ее. Если вы хотите попробовать с неприводимыми многочленами, то GF(2^3) включает следующие элементы: 0, 1, x, $x + 1$, x^2, $x^2 + 1$, $x^2 + x$, $x^2 + x + 1$. Удобный для параллельной реализации алгоритм вычисления обратных значений в GF(2^n) приведен в [421].

При обсуждении полиномов термин "простое число" заменяется термином " неприводимый многочлен". Полином называется неприводимым, если его нельзя представить в виде двух других полиномов (конечно же, кроме 1 и самого полинома). Полином $x^2 + 1$ неприводим над целыми числами, а полином $x^3 + 2x^2 + x$ не является неприводимым, он может быть представлен как $x(x + 1)(x + 1)$.

Полином, который в данном поле является генератором, называется примитивным или базовым, все его коэффициенты взаимно просты. Мы снова вернемся к примитивным полиномам, когда будем говорить о сдвиговых регистрах с линейной обратной связью (см. раздел 16.2).

Вычисления в GF(2^n) могут быть быстро реализованы аппаратно с помощью сдвиговых регистров с линейной обратной связью. По этой причине вычисления над GF(2^n) часто быстрее, чем вычисления над GF(p). Так как возведение в степень в GF(2^n) гораздо эффективнее, то эффективнее и вычисление дискретных логарифмов [180, 181, 368, 379]. Дополнительную информацию об этом можно найти в [140].

Для поля Галуа GF(2^n) криптографы любят использовать в качестве модулей трехчлены $p(x) = x^n + x + 1$, так как длинная строка нулей между коэффициентами при x^n и x позволяет просто реализовать быстрое умножение по модулю [183]. Полином должен быть примитивным, в противном случае математика не будет работать. $x^n + x + 1$ примитивен для следующих значений n, меньших чем 1000 [1649, 1648]:

1, 3, 4, 6, 9, 15, 22, 28, 30, 46, 60, 63, 127, 153, 172, 303, 471, 532, 865, 900

Существуют аппаратные реализации GF(2^{127}), где $p(x) = x^{127} + x + 1$ [1631, 1632, 1129]. Эффективная архитектура аппаратуры возведения в степень для GF(2^n) рассматривается в [147].

11.4 Разложение на множители

Разложить число на множители - значит найти его простые сомножители.

10 = 2*5

60 = 2*2*3*5

252601 = 41*61*101

2113- 1 =3391*23279*65993*1868569*1066818132868207

Разложение на множители является одной из древнейших проблем теории чисел. Этот процесс несложен, но требует времени. Это пока остается так, но ряд сдвигов в этом искусстве все же произошел. Сегодня самым лучшим алгоритмом является:

Решето числового поля чисел (Number field sieve, **NFS**) [953] (см. также [952, 16, 279]). **Решето общего числового поля** - это самый быстрый из известных алгоритм для чисел размером 110 и более разрядов [472, 635]. В своем первоначальном виде он был непрактичен, но за последние несколько лет он был последовательно улучшен [953]. NFS все еще слишком нов, чтобы бить рекорды разложения на множители, но скоро все перем енится. Ранняя версия использовалась для разложения на множители девятого числа Ферма: 2512 + 1 [955,954].

Другие алгоритмы, вытесненные NFS:

Квадратичное решето (Quadratic sieve, **QS**) [1257, 1617, 1259]. Это самый быстрый из известных и чаще всего использовавшийся алгоритм для чисел, длина которых меньше 110 десятичных разрядов [440]. Более б ыстрая версия этого алгоритма называется множественным полиномиальным квадратичным решетом [1453, 302]. Самая быстрая версия называется двойной вариацией множественного полиномиального квадратичного решета с большим простым числом.

Метод эллиптической кривой (Elliptic curve method, **ECM**) [957, 1112, 1113]. Этот метод использовался для поиска не более, чем 43-разрядных множителей.

Алгоритм Монте-Карло Полларда (Pollard's Monte Carlo algorithm) [1254, 248]. (Этот алгоритм также приведен у Кнута в томе 2 [863].)

Алгоритм непрерывных дробей (Continued fraction algorithm). См. [1123, 1252, 863]. Этот алгоритм не подходит по времени выполнения.

Проверка делением (Trial division). Этот самый старый алгоритм разложения на множители состоит из проверки каждого простого числа, меньшего или равного ква дратному корню из раскладываемого числа.

В качестве хорошего введения в различные алгоритмы разложения на множители, кроме NFS, можно и с пользовать [251]. NFS лучше всего рассмотрен в [953]. Более старыми рпаботами являются [505, 1602, 1258]. Сведения о параллельном разложении на множители можно найти в [250].

Если число n на множители раскладывается, то эвристическое время выполнения самых быстрых вариантов QS асимптотически равно:

$$e^{(1+O(1))(\ln(n))^{1/2}(\ln((\ln(n)))^{1/2}}$$

NFS намного быстрее, оценка его эвристического времени выполнения:

$$e^{(1.923+O(1))(\ln(n))^{1/3}(\ln((\ln(n)))^{2/3}}$$

В 1970 году большой новостью стало разложение на множители 41-разрядного трудного числа [1123]. ("Трудным" является такое число, у которого нет маленьких множителей, и которое не обладает специальной формой, позволяющей упростить процесс.) Десять лет спустя разложение в два раз более длинного числа заняло лишь несколько часов на компьютере Cray [440].

В 1988 году Карл Померанс (Carl Pomerance), используя обычные СБИС, спроектировал устройство для ра зложения на множители [1259]. Размер числа, которое можно было разложить, зависел только от размеров ус тройства, которое так и не было построено.

В 1993 году с помощью квадратичного решета было разложено на множители 120-разрядное трудное число. Расчет, потребовавший 825 mips-лет, был выполнен за три месяца реального времени [463]. Другие результаты приведены в [504].

Сегодня для разложения на множители используются компьютерные сети [302, 955]. Для разложения 116_разрядного числа Аржат Ленстра (Arjen Lenstra) и Марк Манасс (Mark Manasse) в течение нескольких м есяцев использовали свободное время массива компьютеров, разбросанных по всему миру, - 400 mips-лет.

В марте 1994 года с помощью двойной вариации множественного полиномиального QS [66] командой м атематиков под руководством Ленстры было разложено на множители 129-разрядное (428-битовое) число. В ычисления выполнялись добровольцами в Internet - в течение восьми месяцев трудились 600 человек и 1600 ко мпьютеров, возможно, самый большой в истории многопроцессорный конгломерат. Трудоемкость вычислений была в диапазоне от 4000 до 6000 mips-лет. Компьютеры соединялись по электронной почте, передавая свои результаты в центральное хранилище, где выполнялся окончательный анализ. В этих вычислениях использов ались QS и теория пятилетней давности, NFS мог бы ускорить выполнение расчетов раз в десять [949]. В соо тветствии с [66]: "Мы делаем вывод, что широко используемые 512-битовые модули RSA могут быть вскрыты организацией, готовой потратить несколько миллионов долларов и подождать несколько месяцев." По оценкам авторов разложение 512-битового числа в 100 раз более трудоемко при использовании той же техники и только в 10 сложнее при использовании NFS и современной техники [949].

С целью развития искусства разложения на множители RSA Data Security, Inc. в марте 1991 года объявило о

программе RSA Factoring Challenge (состязание RSA по разложению на множители) [532]. Состязание состоит в разложении на множители ряда трудных чисел, каждое из которых является произведением двух простых чисел примерно одинакового размера. Каждое простое число было выбрано конгруэнтным 2 по модулю 3. Всего было предложено 42 числа, по одному числу в диапазоне от 100 до 500 разрядов с шагом 10 разрядов (плюс одно д о-полнительное, 129-разрядное число). К моменту написания этой книги RSA-100, RSA-110, RSA-120, и RSA-129 были разложены на множители, все с помощью QS. Следующим (с помощью NFS) может быть RSA-130, или чемпионы по разложению на множители сразу возьмутся за RSA -140.

Данная область развивается быстро. Технику разложения на множители трудно экстраполировать, так как невозможно предсказать развитие математической теории. До открытия NFS многие считали, что любой метод разложения на множители не может асимптотически быть быстрее QS. Они были неправы.

Предстоящее развитие NFS, по видимому, будет происходить в форме уменьшения константы: 1.923. Для ряда чисел специальной формы, таких как числа Ферма, константа приближается к 1.5 [955, 954]. Если бы для трудных чисел, используемых в сегодняшней криптографии, константу тоже можно было снизить до этого уровня, то 1024-битовые числа раскладывались бы на множители уже сегодня. Одним из способов уменьшить константу является обнаружение лучших способов представления чисел как полиномов с маленькими коэфф и-циентами. Пока еще проблема не изучалась достаточно эффективно, но возможно решающий успех уже близок [949].

Последние результаты программы RSA Factoring Challenge можно узнать, отправив запрос по электронной почте по адресу challenge-info@rsa.com.

Квадратные корни по модулю n

Если n - произведение двух простых чисел, то возможность вычислить квадратные корни по модулю n вычислительно эквивалентна возможности разложить число n на множители [1283, 35, 36, 193]. Другими словами, тот, кто знает простые множители числа n, может легко вычислить квадратные корни любого числа по модулю n, но для любого другого вычисление окажется таким же трудным, как и разложение на простые множители числа n.

11.5 Генерация простого числа

Для алгоритмов с открытыми ключами нужны простые числа. Их нужно множество для любой достаточно большой сети. Прежде, чем обсуждать математику генерации простого числа, я отвечу на несколько очевидных вопросов.

Если каждому понадобится свое простое число, не иссякнет ли у нас запас? Нет. В действительности сущес т-вует приблизительно 10151 простых чисел длино1 до 512 бит включительно. Для чисел, близких n, вероятность того, что случайно выбранное число окажется простым, равна $1/\ln n$. Поэтому полное число простых чисел, меньших n, равно $n/(\ln n)$. Во вселенной всего 10^{77} атомов. Если бы для каждого атома во вселенной с начала времен каждую микросекунду требовался бы миллиард простых чисел, понадобилось бы только 10^{109} простых чисел, осталось бы еще примерно 10^{151} простых чисел.

Что если два человека случайно выберут одно и то же простое число? Этого не случится. При выборе из 10151 простых чисел вероятность совпадения выбора значительно меньше, чем вероятность, что ваш компь ю-тер случайно вспыхнет в тот самый момент, когда вы выиграете в лотерею.

Если кто-то создаст базу данных всех простых чисел, не сможет ли он использовать эту базу данных для вскрытия алгоритмов с открытыми ключами? Нет. Если бы вы хранили один гигабайт информации на устро й-стве, весящем один грамм, то перечень простых чисел размером до 512 бит включительно весил бы столько, что масса хранилища превысила бы предел Чандрасекара, и оно сколлапсировало бы в черную дыру ... в любом случае вы не сможете извлечь данные.

Но если так трудоемко разложение на множители, как может быть простой генерация простых чисел? Фокус в том, что ответить "да" или "нет" на вопрос "Является ли число n простым?" гораздо проще, чем ответить на более сложный вопрос "Каковы множители n?"

Генерация случайных чисел с последующей попыткой разложения их на множители - это неправильный сп о-соб поиска простых чисел. Существуют различные вероятностные проверки на простоту чисел, определяющие, является ли число простым, с заданной степенью достоверности. При условии, что эта "степень достоверности" достаточна велика, такие способы проверки достаточно хороши. Я слышал, что простые числа, генерированные таким образом называются "промышленно простыми числами": эти числа вероятно являются простыми с ко нтролируемой возможностью ошибки.

Предположим, что одна проверка из 2^{50} - ошибочна. Это означает, что с вероятностью $1/10^{15}$ проверка объя-вит простым составное число. (Простое число никогда не будет объявлено составным при проверке.) Если по

какой-то причине понадобится большая достоверность простоты числа, уровень ошибки можно понизить. С другой стороны, если вы установите вероятность того, что число является составным, в 300 миллионов раз меньшей, чем вероятность выиграть главный приз в государственной лотерее, вы можете больше об этом не волноваться.

Обзоры недавних исследований в этой области можно найти в [1256, 206]. Другими важными работами я вляются [1490, 384, 11, 19, 626, 651, 911].

Solovay-Strassen

Роберт Соловэй (Robert Solovay) и Фолькер Штрассен (Volker Strassen) разработали алгоритм вероятностной проверки простоты числа [1490]. Для проверки простоты числа p этот алгоритм использует символ Якоби:

(1) Выберите случайно число a, меньшее p.

(2) Если НОД(a,p) (1, то p не проходит проверку и является составным.

(3) Вычислите $j = a(p\text{-}1)/2 \bmod p$.

(4) Вычислите символ Якоби J(a,p).

(5) Если $j \neq$ J(a,p), то число p наверняка не является простым.

(6) Если $j =$ J(a,p), то вероятность того, что число p не является простым, не больше 50 процентов.

Число a, которое не показывает, что p наверняка не является простым числом, называется свидетелем. Если p - составное число, вероятность случайного числа a быть свидетелем не ниже 50 процентов. Повторите эту проверку t раз с t различными значениями a. Вероятность того, что составное число преодолеет все t проверок, не превышает $1/2^t$.

Lehmann

Другой, более простой тест был независимо разработан Леманном (Lehmann) [903]. Вот последовательность действий при проверке простоты числа p:

(1) Выберите случайно число a, меньшее p.

(2) Вычислите $a^{(p\text{-}1)/2} \bmod p$.

(3) Если $a^{(p\text{-}1)/2} \neq 1$ или -1 (mod p), то p не является простым.

(4) Если $a^{(p\text{-}1)/2} \equiv 1$ или -1 (mod p), то вероятность того, что число p не является простым, не больше 50 процентов.

И снова, вероятность того, что случайное число a будет свидетелем составной природы числа p, не меньше 50 процентов. Повторите эту проверку t раз. Если результат вычислений равен 1 или -1, но не всегда равен 1, то p является простым числом с вероятностью ошибки $1/^{2t}$.

Rabin-Miller

Повсеместно используемым является простой алгоритм, разработанный Майклом Рабином (Michael Rabin), частично основанным на идеях Гэри Миллера [1093, 1284]. По сути, это упрощенная версия алгоритма, рек омендованного в предложении DSS proposal [1149, 1154].

Выберите для проверки случайное число p. Вычислите b - число делений p - 1 на 2 (т.е., 2^b - это наибольшая степень числа 2, на которое делится p - 1). Затем вычислите m, такое что $p = 1 + 2^b * m$.

(1) Выберите случайное число a, меньшее p.

(2) Установите $j = 0$ и $z = am \bmod p$.

(3) Если $z = 1$ или если $z = p$ - 1, то p проходит проверку и может быть простым числом.

(4) Если $j > 0$ и $z = 1$, то p не является простым числом.

(5) Установите $j = j + 1$. Если $j < b$ и $z($ p - 1, установите $z = z^2 \bmod p$ и вернитесь на этап (4). Если $z = p$ - 1, то p проходит проверку и может быть простым числом.

(6) Если $j = b$ и $z \neq p$ - 1, то p не является простым числом.

В этом тесте вероятность прохождения проверки составным числом убывает быстрее, чем в предыдущих. Гарантируется, что три четверти возможных значений a окажутся свидетелями. Это означает, что составное число проскользнет через t проверок с вероятностью не большей $(1/4)^t$, где t - это число итераций. На самом деле и эти оценки слишком пессимистичны. Для большинства случайных чисел около 99.9 процентов возмо ж-

ных значений a являются свидетелями [96].

Существуют более точные оценки [417]. Для n-битового кандидата в простые числа (где n больше 100), вероятность ошибки в одном тесте меньше, чем $4n2^{(k/2)^{1/2}}$. И для 256-битового n вероятность ошибки в шести тестах меньше, чем $1/2^{51}$. Дополнительную теорию можно найти в [418].

Практические соображения

В реальных приложениях генерация простых чисел происходит быстро.

(1) Сгенерируйте случайное n-битовое число p.

(2) Установите старший и младший биты равными 1. (Старший бит гарантирует требуемую длину простого числа, а младший бит обеспечивает его нечетность.)

(3) Убедитесь, что p не делится на небольшие простые числа: 3, 5, 7, 11, и т.д. Во многих реализациях проверяется делимость p на все простые числа, меньшие 256. Наиболее эффективной является проверка на делимость для всех простых чисел, меньших 2000 [949]. Это может быть эффективно выполнено с помощью колеса [863].

(4) Выполните тест Rabin-Miller для некоторого случайного a. Если p проходит тест, сгенерируйте другое случайное a и повторите проверку. Выбирайте небольшие значения a для ускорения вычислений. Выполните пять тестов [651]. (Одного может показаться достаточным, но выполните пять.) Если p не проходит одной из проверок, сгенерируйте другое p и попробуйте снова.

Иначе, можно не генерировать p случайным образом каждый раз, но последовательно перебирать числа, начиная со случайно выбранного до тех пор, пока не будет найдено простое число.

Этап (3) не является обязательным, но это хорошая идея. Проверка, что случайное нечетное p не делится на 3, 5 и 7 отсекает 54 процента нечетных чисел еще до этапа (4). Проверка делимости на все простые числа, меньшие 100, убирает 76 процентов нечетных чисел, проверка делимости на все простые числа, меньшие 256, убирает 80 процентов нечетных чисел. В общем случае, доля нечетных кандидатов, которые не делятся ни на одно простое число, меньшее n, равна $1.12/\ln n$. Чем больше проверяемое n, тем больше предварительных вычислений нужно выполнить до теста Rabin-Miller.

Одна из реализаций этого метода на Sparc II способна находить 256-битовые простые числа в среднем за 2.8 секунды, 512-битовые простые числа - в среднем за 24.0 секунды, 768-битовые простые числа - в среднем за 2.0 минуты, а 1024-битовые простые числа - в среднем за 5.1 минуты [918].

Сильные простые числа

Если n - произведение двух простых чисел, p и q, то может понадобиться использовать в качестве p и q **сильные простые числа**. Такие простые числа обладают рядом свойств, которые усложняют разложение произведения n определенными методами разложения на множители. Среди таких свойств были предложены [1328, 651]:

Наибольший общий делитель $p - 1$ и $q - 1$ должен быть небольшим.

И $p - 1$, и $q - 1$ должны иметь среди своих множителей большие простые числа, соответственно p' и q'.

И $p' - 1$, и $q' - 1$ должны иметь среди своих множителей большие простые числа.

И $p + 1$, и $q + 1$ должны иметь среди своих множителей большие простые числа.

И $(p - 1)/2$, и $(q - 1)/2$ должны быть простыми [182]. (Обратите внимание, при выполнении этого условия выполняются и два первых.)

Насколько существенно применение именно сильных простых чисел, остается предметом продолжающихся споров. Эти свойства были разработаны, чтобы затруднить выполнение ряда старых алгоритмов разложения на множители. Однако самые быстрые алгоритмы одинаково быстры при разложении на множители любых чисел, как удовлетворяющих приведенным условиям, так и нет [831].

Я против специальной генерации сильных простых чисел. Длина простых чисел гораздо важнее их структуры. Более того, сама структура уменьшает случайность числа и может снизить устойчивость системы.

Но все может измениться. Могут быть созданы новые методы разложения на множители, которые лучше работают с числами, обладающими определенными свойствами. В этом случае снова могут потребоваться сильные простые числа. Заглядывайте в журналы по теоретической математике.

11.6 Дискретные логарифмы в конечном поле

В качестве другой однонаправленной функции в криптографии часто используется возведение в степень по модулю. Легко вычислить:

$a^x \bmod n$

Задачей, обратной возведению в степень по модулю, является поиск дискретного логарифма. А это уже н елегкая задача:

Найти x, для которого $a^x \equiv b \pmod n$.

Например:

Если $3^x \equiv 15 \bmod 17$, то $x = 6$

Решения существуют не для всех дискретных логарифмов (помните, речь идет только о целочисленных р ешениях). Легко заметить, что следующее уравнение не имеет решений

$3^x \equiv 7 \pmod{13}$

Еще сложнее решать эту задачу для 1024-битовых чисел.

Вычисление дискретных логарифмов в конечной группе

Криптографы интересуются дискретными логарифмами следующих трех групп:

— Мультипликативная группа полей простых чисел: GF(p)

— Мультипликативная группа конечных полей степеней 2: GF(2^n)

— Группы эллиптической кривой над конечными полями F: EC(F)

Безопасность многих алгоритмов с открытыми ключами основана на задаче поиска дискретных логарифмов, поэтому эта задача была глубоко изучена. Хороший подробный обзор этой проблемы и ее наилучшие решения на соответствующий момент времени можно найти в [1189, 1039]. Лучшей современной статьей на эту тему является [934].

Если p является простым числом и используется в качестве модуля, то сложность поиска дискретных лог арифмов в GF(p) по существу соответствует разложению на множители числа n того же размера, где n - это произведение двух простых чисел приблизительно равной длины [1378,934]. То есть:

$$e^{(1+O(1))(\ln(n))^{1/2}(\ln((\ln(n)))^{1/2}}$$

Решето числового поля быстрее, оценка его эвристического времени выполнения:

$$e^{(1.923+O(1))(\ln(n))^{1/3}(\ln((\ln(n)))^{2/3}}$$

Стивен Полиг (Stephen Pohlig) и Мартин Хеллман нашли способ быстрого вычисления дискретных лог арифмов в GF(p) при условии, что p - 1 раскладывается на малые простые множители [1253]. По этой причине в криптографии используются только такие поля, для которых p - 1 обладает хотя бы одним большим простым множителем. Другой алгоритм [14] вычисляет дискретных логарифм со скоростью, сравнимой с разложением на множители, он был расширен на поля вида GF(p^n) [716]. Этот алгоритм был подвергнут критике в [727] по ряду теоретических моментов. В других статьях [1588] можно увидеть, насколько на самом деле трудна пр облема в целом.

Вычисление дискретных логарифмов тесно связано с разложением на множители. Если вы можете решить проблему дискретного логарифма, то вы можете и разложить на множители. (Истинность обратного никогда не была доказана.) В настоящее время существует три метода вычисления дискретных логарифмов в поле простого числа [370, 934, 648]: линейное решето, схема целых чисел Гаусса и решето числового поля.

Предварительное, объемное вычисление для поля должно быть выполнено только один раз. Затем, быстро можно вычислять отдельные логарифмы. Это может серьезно уменьшить безопасность систем, основанных на таких полях. Важно, чтобы различные приложения использовали различные поля простых чисел. Хотя нескол ько пользователей одного приложения могут применять общее поле.

В мире расширенных полей исследователями не игнорируются и GF(2^n). Алгоритм был предложен в [727]. Алгоритм Копперсмита (Coppersmith) позволяет за приемлемое время находить дискретные логарифмы в таких полях как GF(2^{127}) и делает принципиально возможным их поиск в полях порядка GF(2^{400}) [368]. В его основе лежит [180]. У этого алгоритма очень велика стадия предварительных вычислений, но во всем остальном он хорош и эффективен. Реализация менее эффективной версии этого же алгоритма после семи часов предвар ительных вычислений тратила на нахождение каждого дискретного логарифма в поле GF(2^{127}) лишь несколько

секунд [1130, 180]. (Это конкретное поле, когда-то использовавшееся в некоторых криптосистемах [142, 1631, 1632], не является безопасным.) Обзор некоторых из этих результатов можно найти в [1189, 1039].

Позднее были выполнены предварительные вычисления для полей GF(2^{227}), GF(2^{313}) и GF(2^{401}), удалось значительно продвинуться и для поля GF(2^{503}). Эти вычисления проводились на nCube-2, массивном параллельном компьютере с 1024 процессорами [649, 650]. Вычисление дискретных логарифмов в поле GF(2^{593}) все еще находится за пределами возможного.

Как и для нахождения дискретных логарифмов в поле простого числа, для вычисления дискретных логарифмов в полиномиальном поле также требуется один раз выполнить предварительные вычисления. Тахер Эль-Джамаль (Taher ElGamal) [520] приводит алгоритм вычисления дискретных логарифмов в поле GF(p^2).

Глава 12 Стандарт шифрования данных DES (Data Encryption Standard)

12.1 Введение

Стандарт шифрования данных DES (Data Encryption Standard), который ANSI называет Алгоритмом ши фрования данных DEA (Data Encryption Algorithm), а ISO - DEA-1, за 20 лет стал мировым стандартом. Хотя на нем и появился налет старости, он весьма прилично выдержал годы криптоанализа и все еще остается безопа сным по отношению ко всем врагам, кроме, возможно, самых могущественных.

Разработка стандарта

В начале 70-х годов невоенные криптографические исследования были крайне редки. В этой области почти не публиковалось исследовательских работ. Большинство людей знали, что для своих коммуникаций военные используют специальную аппаратуру кодирования, но мало кто разбирался в криптографии как в науке. Заме тными знаниями обладало Агентство национальной безопасности (National Security Agency, NSA), но оно даже не признавало публично своего собственного существования.

Покупатели не знали, что они покупают. Многие небольшие компании изготавливали и продавали крипт ографическое оборудование, преимущественно заокеанским правительствам. Все это оборудование отличалось друг от друга и не могло взаимодействовать. Никто не знал, действительно ли какое-либо из этих устройств безопасно, не существовало независимой организации, которая засвидетельствовала бы безопасность. Как гов орилось в одном из правительственных докладов [441]:

> Влияние соответствующего изменения ключей и принципов работы на реальную мощь аппаратуры шифров а ния/дешифрирования было (и фактически осталось) неизвестным почти всем покупателям, и было очень трудно принимать обоснованные решения о генерации ключей, правильном диалоговом или автономном режиме, и т.д., которые отвечали бы потребностям покупателей в безопасности.

В 1972 году Национальное бюро стандартов (National Bureau of Standards, NBS), теперь называющееся Н ациональным институтом стандартов и техники (National Institute of Standards and Technology, NIST), выступило инициатором программы защиты линий связи и компьютерных данных. Одной из целей этой программы была разработка единого, стандартного криптографического алгоритма. Этот алгоритм мог бы быть проверен и се ртифицирован, а использующие его различные криптографические устройства могли бы взаимодействовать. Он мог бы, к тому же, быть относ ительно недорогим и легко доступным.

15 мая 1973 года в *Federal Register* NBS опубликовало требования к криптографическому алгоритму, кот орый мог бы быть принят в качестве стандарта. Было пр иведено несколько критериев оценки проекта:

— Алгоритм должен обеспечивать высокий уровень безопасности.

— Алгоритм должен быть полностью определен и легко понятен.

— Безопасность алгоритма должна основываться на ключе и не должна зависеть от сохранения в тайне с амого алгоритма.

— Алгоритм должен быть доступен всем пользователям.

— Алгоритм должен позволять адаптацию к различным применениям.

— Алгоритм должен позволять экономичную реализацию в виде электронных приборов.

— Алгоритм должен быть эффективным в использовании.

— Алгоритм должен предоставлять возможности проверки.

— Алгоритм должен быть разрешен для экспорта.

Реакция общественности показала, что к криптографическому стандарту существует заметный интерес, но опыт в этой области чрезвычайно мал. Ни одно из предложений не удовлетворяло предъявленным требованиям.

27 августа 1972 года в *Federal Register* NBS опубликовало повторное предложение. Наконец, у Бюро появи лся подходящий кандидат: алгоритм под именем Люцифер, в основе которого лежала разработка компании IBM, выполненная в начале 70-х (см. раздел 13.1). В IBM существовала целая команда криптографов, работавшая в Кингстоне (Kingston) и Йорктаун Хайтс (Yorktown Heights), в которую входили Рой Адлер (Roy Adler), Дон Копперсмит (Don Coppersmith), Хорст Фейстель (Horst Feistel), Эдна Кроссман (Edna Crossman), Алан Конхейм (Alan Konheim), Карл Майер (Carl Meyer), Билл Ноц (Bill Notz), Линн Смит (Lynn Smith), Уолт Тачмен (Walt Tuchman) и Брайант Такерман (Bryant Tuckerman).

Несмотря на определенную сложность алгоритм был прямолинеен. Он использовал только простые логич е-

ские операции над небольшими группами битов и мог быть довольно эффективно реализован в аппаратуре.

NBS попросило NSA помочь оценить безопасность алгоритма и определить, подходит ли он для использов ания в качестве федерального стандарта. IBM уже получила патент [514], но желала сделать свою интеллект уальную собственность доступной для производства, реализации и использования другими компаниями. В конце концов, NBS и IBM выработали соглашение, по которому NBS получало неисключительную, бесплатную л ицензию изготавливать, использовать и продавать устройства, реализующие этот алгоритм.

Наконец, 17 марта 1975 года в *Federal Register* NBS опубликовало и подробности алгоритма, и заявление IBM о предоставлении неисключительной, бесплатной лицензии на алгоритм, а также предложило присылать комментарии по поводу данного алгоритма [536]. В другой заметке в *Federal Register*, 1 августа 1975 года, различным организациям и широкой публике снова предлагалось прокомментировать предложенный алгоритм.

И комментарии появились [721, 497, 1120). Многие настороженно относились к участию "невидимой руки" NSA в разработке алгоритма. Боялись, что NSA изменит алгоритм, вставив в него потайную дверцу. Жалов ались, что NSA уменьшило длину ключей с первоначальных 128 битов до 56 (см. раздел 13.1). Жаловались на внутренние режимы работы алгоритма. Многие соображения NSA стали ясны и понятны в начале 90-х, но в 70-х они казались таинственными и тревожными.

В 1976 году NBS провело два симпозиума по оценке предложенного стандарта. На первом обсуждались м атематика алгоритма и возможность потайной дверцы [1139]. На втором - возможности увеличения длины ключа алгоритма [229]. Были приглашены создатели алгоритма, люди, оценивавшие алгоритм, разработчики аппар атуры, поставщики, пользователи и критики. По всем отчетам симпозиумы были весьма ожи вленными [1118].

Несмотря на критику Стандарт шифрования данных DES 23 ноября 1976 года был принят в качестве фед ерального стандарта [229] и разрешен к использованию на всех несекретных правительственных коммуникац иях. Официальное описание стандарта, FIPS PUB 46, "Data Encryption Standard", было опубликовано 15 января 1977 года и вступило в действие шестью месяцами позже [1140]. FIPS PUB 81, " Modes of DES Operation" (Режимы работы DES), было опубликовано в 1980 году [1143]. FIPS PUB 74, "Guidelines for Implementing and Using the NBS Data Encryption Standard" (Руководство по реализации и использованию Стандарта шифрования данных NBS), появилось в 1981 году [1142]. NBS также опубликовало FIPS PUB 112, специфицируя DES для шифрования паролей [1144], и FIPS PUB 113, специфицируя DES для проверки подлинности компьютерных данных [1145]. (FIPS обозначает Federal Information Processing Standard.)

Эти стандарты были беспрецедентными. Никогда до этого оцененный NSA алгоритм не был опубликован. Возможно эта публикация была следствием непонимания, возникшего между NSA и NBS. NSA считало, что DES будет реализовываться только аппаратно. В стандарте требовалась именно аппаратная реализация, но NBS опубликовало достаточно информации, чтобы можно было создать и программную реализацию DES. Не для печати NSA охарактеризовало DES как одну из своих самых больших ошибок. Если бы Агентство предполаг ало, что раскрытые детали позволят писать программное обеспечение, оно никогда бы не согласилось на это. Для оживления криптоанализа DES сделал больше, чем что-либо другое. Теперь для исследования был доступен алгоритм, который NSA объявило безопасным. Не случайно следующий правительственный стандарт алгори тма, Skipjack (см. раздел 13.12.), был засекречен.

Принятие стандарта

Американский национальный институт стандартов (American National Standards Institute, ANSI) одобрил DES в качестве стандарта для частного сектора в 1981 году (ANSI X3.92.) [50], назвав его Алгоритмом шифр ования данных (Data Encryption Algorithm, DEA). ANSI опубликовал стандарт режимов работы DEA (ANSI X3.106) [52], похожий на документ NBS, и стандарт для шифрования в сети, использующий DES (ANSI X3.105) [51].

Две другие группы внутри ANSI, представляющие банковские операции при розничной и оптовой торговле, разработали свои стандарты на основе DES. Банковские операции при розничной торговле включают транза кции между финансовыми организациями и отдельными личностями, а банковские операции при оптовой то рговле включают транзакции между финансовыми организациями.

Рабочая группа ANSI по безопасности финансовых организаций при розничной торговле разработала ста ндарт для управления PIN-кодами и их безопасностью (ANSI X9.8) [53] и другой использующий DES стандарт для проверки подлинности финансовых сообщений о розничных продажах (ANSI X9.19) [56]. Эта группа ра зработала и проект стандарта для безопасного распределения ключей (ANSI X9.2.4) [58].

Рабочая группа ANSI по безопасности финансовых организаций при оптовой торговле разработала свой со бственный набор стандартов для проверки подлинности сообщений (ANSI X9.9) [54], управления ключами (ANSIX9.17) [55, 1151], шифрования (ANSIX9.2.3) [57] и безопасной проверки подлинности личностей и узлов (ANSI X9.26) [59].

Американская ассоциация банкиров разрабатывает необязательные стандарты для финансовой индустрии.

Они опубликовали стандарт, рекомендующий DES для шифрования [1], и другой стандарт для управления криптографическими ключами [2].

До появления в 1987 году Акта о компьютерной безопасности (Computer Security Act) the за разработку ф едеральных стандартов в области телекоммуникаций отвечала Администрация общих служб (General Services Administration, CSA), а с этого момента ответственность перешла к NIST. CSA опубликовала три стандарта, использующих DES: два для требований к общей безопасности и возможности взаимодействия (Федеральный стандарт 1026 [662] и Федеральный стандарт 1027 [663]) и один для факс-аппаратов Group 3 (Федеральный стандарт 1028 [664]).

Казначейство издало стратегические директивы, требующие, чтобы подлинность всех сообщений о переводе электронных финансов удостоверялась с помощью DES [468, 470]. Оно также разработало основанный на DES критерий, которому должны удовлетворять все устройства проверки подлинности [469].

ISO сначала проголосовала за введение DES, называемого в ее интерпретации DEA-1, в качестве междун ародного стандарта, а затем приняла решение не заниматься стандартизацией криптографии. Однако в 1987 году группа ISO, занимающаяся международными стандартами в области оптовой торговли, применила DES в ме ждународном стандарте проверки подлинности [758] и для управления ключами [761]. DES также используется в качестве австралийского банковского стандарта [1497].

Проверка и сертификация оборудования DES

Частью стандарта DES является проверка NIST реализаций DES. Эта проверка подтверждает, что реализ ация соответствует стандарту. До 1994 года NIST проверял только аппаратные и программно-аппаратные реал изации - пока стандарт запрещал программные реализации. На март 1995 года 73 различных реализации были признаны соответствующими стандарту.

NIST также разработал программу сертификации устройств проверки подлинности на соответствие ANSI X9.9 и FIPS 113. На март 1995 года было сертифицировано 33 различных продукта. Казначейство использует свою собственную дополнительную процедуру сертификации. У NIST также есть программа проверки аппар атуры на соответствие ANSI X9.17 для управления ключами при оптовой торговле [1151], На март 1995 года было сертифицировано четыре продукта.

1987

В стандарте DES было оговорено, что он будет пересматриваться каждые пять лет. В 1983 DES был повто рно сертифицирован без всяких проблем. 6 марта 1987 года в Federal Register NBS попросило прокомментир овать предложение на следующие пять лет. NBS предложило на обсуждение следующие три альтернативы [1480, 1481]: вновь подтвердить стандарт на следующие пять лет, отказаться от стандарта или пересмотреть примен имость стандарта.

NBS и NSA пересмотрели стандарт. В этот раз NSA было задействовано в большей степени. Благодаря по дписанной Рейганом директиве NSDD-145 NSA получило право вето по отношению к деятельности NBS в обла сти криптографии. Первоначально NSA объявило, что оно не сертифицирует стандарт повторно. Проблема была не в том, что DES действительно был взломан, и даже не в том, что он, может быть, был взломан. По видим ому, предполагалось, что он вот-вот будет взломан.

Само по себе NSA предложило Программу коммерческой подписи COMSEC (Commercial COMSEC Endorsement Program, CCEP), которая по сути представляла собой набор алгоритмов для замены DES [85]. Эти разработанные NSA алгоритмы не были опубликованы и были доступны только в виде защищенных от взлома СБИС (см. раздел 25.1).

Это предложение не было принято. Было отмечено, что DES широко используется в бизнесе (особенно в ф инансах), и что приемлемой альтернативы не существует. Отказ от стандарта оставил бы многие организации без защиты данных. После длительных споров DES был вновь утвержден в качестве правительственного стандарта США до 1992 года [1141]. NBS решило, что DES никогда больше не будет сертифицирован снова [1480].

1993

Никогда не говори "никогда". В 1992 году альтернативы алгоритму DES все еще не было. NBS, называемый теперь NIST, снова в Federal Register предложило пр окомментировать DES [540]:

> Цель этого предложения состоит в том, чтобы объявить о предстоящем оценивании адекватности стандарта задаче защ иты компьютерных данных на современном уровне. Промышленности и широкой публике предлагаются три следующих вар ианта решения для FIPS 46-1. Комментарии должны содержать стоимость (последствия) и преимущества этих вариа нтов:
>
> —Повторно принять стандарт на следующие пять (5) лет. Национальный институт стандартов и технологии продолжит сертификацию аппаратуры, реализующей стандарт. FIPS 46-1 будет и дальше оставаться единственным признанным м етодом защиты несекретных компьютерных данных.
>
> —Отказаться от стандарта. Национальный институт стандартов и технологии больше не будет поддерживать стандарт.

> Организации могут продолжать использовать существующую аппаратуру, реализующую стандарт. Заменяя DES, NIST издаст другие стандарты.
> —Пересмотреть положения стандарта о применимости и/или провести ревизию реализации. Такая ревизия должна включать изменения стандарта, позволяющие использовать как аппаратные, так программные и реализации DES, и с-пользовать DES итеративно в определенных приложениях, использовать альтернативные алгоритмы, признанные и зар е-гистрированные NIST.

Срок принятия предложений истек 10 декабря 1992 года. Согласно Рэймонду Каммеру (Raymond Kammer), в то время директору NIST [812]:

> В прошлом году NIST формально предложило присылать комментарии по поводу повторной сертификации DES. Ра с-смотрев присланные предложения и другие технические источники, я собираюсь рекомендовать министру торговли, чтобы он повторно сертифицировал DES еще на пять лет. Я также собираюсь предложить министру, чтобы, объявляя о повторной сертификации, мы сформулировали наши намерения рассмотреть в течение этих пяти лет возможные альтернативы. Делая подобное заявление, мы надеемся дать людям возможность высказаться по поводу предстоящих технологических изменений. В то же время, нам нужно учитывать большое количество систем, использующих этот одобренный ста ндарт.

Несмотря на то, что Управление оценки технологий ссылалось на слова работавшего в NIST Денниса Бра н-стида (Dennis Branstead) от том, что полезное время жизни DES закончится в конце 90-х [1191], алгоритм был сертифицирован повторно на следующие пять лет [1150]. Наконец было разрешено сертифицировать и пр о-граммные реализации DES. Хотелось бы знать, что случится в 1998 году?

12.2 Описание DES

DES представляет собой блочный шифр, он шифрует данные 64-битовыми блоками. С одного конца алг о-ритма вводится 64-битовый блок открытого текста, а с другого конца выходит 64-битовый блок шифротекста. DES является симметричным алгоритмом: для шифрования и дешифрирования используются одинаковые алг о-ритм и ключ (за исключением небольших различий в использовании ключа).

Длина ключа равна 56 битам. (Ключ обычно представляется 64-битовым числом, но каждый восьмой бит используется для проверки четности и игнорируется. Биты четности являются наименьшими значащими битами байтов ключа.) Ключ, который может быть любым 56-битовым числом, можно изменить в любой момент вр е-мени. Ряд чисел считаются слабыми ключами, но их можно легко избежать. Безопасность полностью опред е-ляется ключом.

На простейшем уровне алгоритм не представляет ничего большего, чем комбинация двух основных методов шифрования: смещения и диффузии. Фундаментальным строительным блоком DES является применение к те к-сту единичной комбинации этих методов (подстановка, а за ней - перестановка), зависящей от ключа. Такой блок называется этапом. DES состоит из 16 этапов, одинаковая комбинация методов применяется к открытому тексту 16 раз (см. 11-й).

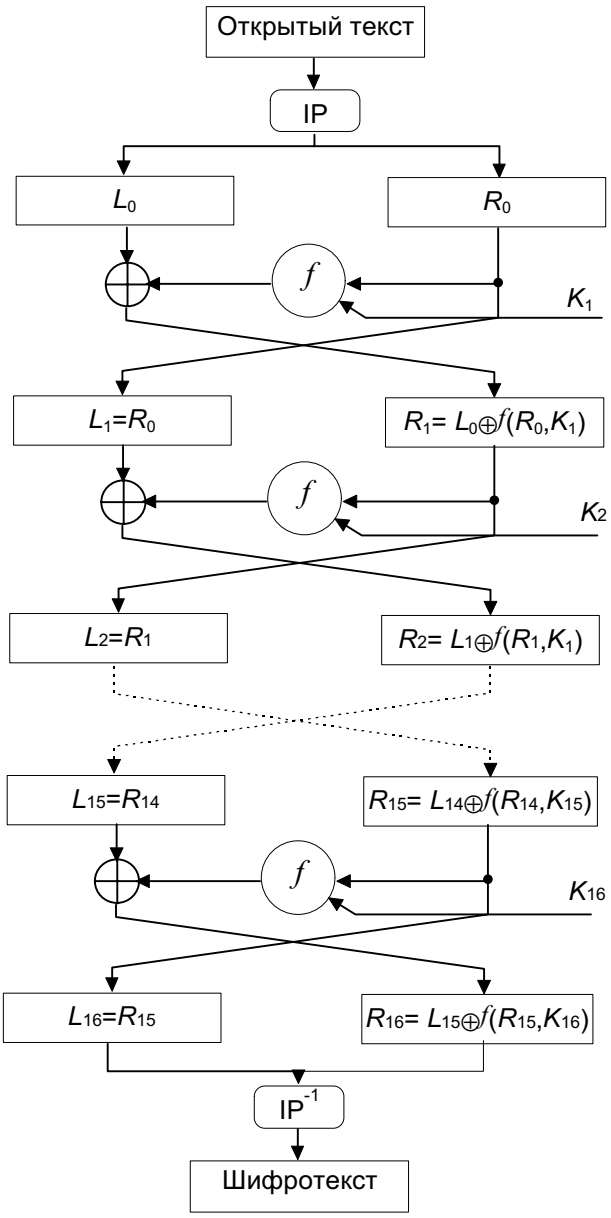


Рис. 12-1. DES.

Алгоритм использует только стандартную арифметику 64-битовых чисел и логические операции, поэтому он легко реализовывался в аппаратуре второй половины 70-х. Изобилие повторений в алгоритме делает его ид еальным для реализации в специализированной микросхеме. Первоначальные программные реализации были довольно неуклюжи, но сегодняшние программы намного лучше.

Схема алгоритма

DES работает с 64-битовым блоком открытого текста. После первоначальной перестановки блок разбивается на правую и левую половины длиной по 32 бита. Затем выполняется 16 этапов одинаковых действий, называ емых функцией f, в которых данные объединяются с ключом. После шестнадцатого этапа правая и левая полов ины объединяются и алгоритм завершается заключительной перестановкой (обратной по отношению к первон ачальной).

На каждом этапе (см. 10-й) биты ключа сдвигаются, и затем из 56 битов ключа выбираются 48 битов. Пр авая половина данных увеличивается до 48 битов с помощью перестановки с расширением, объединяется п осредством XOR с 48 битами смещенного и переставленного ключа, проходит через 8 S-блоков, образуя 32 н овых бита, и переставляется снова. Эти четыре операции и выполняются функцией f. Затем результат функции f объединяется с левой половиной с помощью другого XOR. В итоге этих действий появляется новая правая п оловина, а старая правая половина становится новой левой. Эти действия повторяются 16 раз, образуя 16 этапов DES.

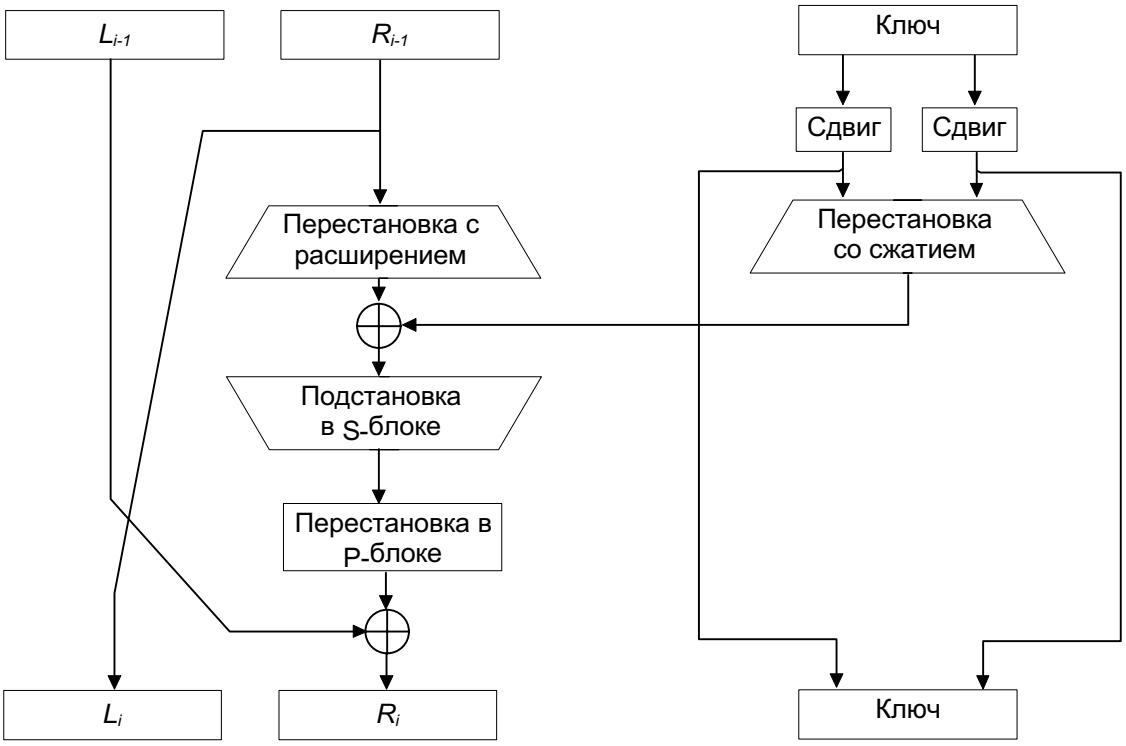


Рис. 12-2. Один этап DES.

Если B_i - это результат i-ой итерации, L_i и R_i - левая и правая половины B_i, K_i - 48-битовый ключ для этапа i, а f - это функция, выполняющие все подстановки, перестановки и XOR с ключом, то этап можно представить как:

$L_i = R_{i-1}$

$R_i = L_{i-1} \oplus \mathrm{f}(R_{i-1}, K_i)$

Начальная перестановка

Начальная перестановка выполняется еще до этапа 1, при этом входной блок переставляется, как показано в 11-й. Эту и все другие таблицы этой главы надо читать слева направо и сверху вниз. Например, начальная пер е-становка перемещает бит 58 в битовую позицию 1, бит 50 - в битовую позицию 2, бит 42 - в битовую позицию 3, и так далее.

Табл. 12-1.
Начальная перестановка

58,	50,	42,	34,	26,	18,	10,	2,	60,	52,	44,	36,	28,	20,	12,	4,
62,	54,	46,	38,	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

Начальная перестановка и соответствующая заключительная перестановка не влияют на безопасность DES. (Как можно легко заметить, эта перестановка в первую очередь служит для облегчения побайтной загрузки да нных открытого текста и шифротекста в микросхему DES. Не забывайте, что DES появился раньше 16- и 32-битовых микропроцессорных шин.) Так как программная реализация этой многобитовой перестановки нелегка (в отличие от тривиальной аппаратной), во многих программных реализациях DES начальная и заключител ьные перестановки не используются. Хотя такой новый алгоритм не менее безопасен, чем DES, он не соответс т-вует стандарту DES и, поэтому, не может называться DES.

Преобразования ключа

Сначала 64-битовый ключ DES уменьшается до 56-битового ключа отбрасыванием каждого восьмого бита, как показано в 10-й. Эти биты используются только для контроля четности, позволяя проверять правильность ключа. После извлечения 56-битового ключа для каждого из 16 этапов DES генерируется новый 48-битовый

подключ. Эти подключи, K_i, определяются следующим образом.

Табл. 12-2.
Перестановка ключа

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,
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

Во первых, 56-битовый ключ делится на две 28-битовых половинки. Затем, половинки циклически сдвигаются налево на один или два бита в зависимости от этапа. Этот сдвиг показан в 9-й.

Табл. 12-3.
Число битов сдвига ключа в зависимости от этапа

Этап	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Число	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

После сдвига выбирается 48 из 56 битов. Так как при этом не только выбирается подмножество битов, но и изменяется их порядок, эта операция называется **перестановка со сжатием**. Ее результатом является набор из 48 битов. Перестановка со сжатием (также называемая переставленным выбором) определена в 8-й. Например, бит сдвинутого ключа в позиции 33 перемещается в позицию 35 результата, а 18-й бит сдвинутого ключа отбрасывается.

Табл. 12-4.
Перестановка со сжатием

14,	17,	11,	2,4,	1,	5,	3,	28,	15,	6,	21,	10,
23,	19,	11,	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

Из-за сдвига для каждого подключа используется отличное подмножество битов ключа. Каждый бит используется приблизительно в 14 из 16 подключей, хотя не все биты используются в точности одинаковое число раз.

Перестановка с расширением

Эта операция расширяет правую половину данных, R_i, от 32 до 48 битов. Так как при этом не просто повторяются определенные биты, но и изменяется их порядок, эта операция называется **перестановкой с расширением**. У нее две задачи: привести размер правой половины в соответствие с ключом для операции XOR и получить более длинный результат, который можно будет сжать в ходе операции подстановки. Однако главный криптографический смысл совсем в другом. За счет влияния одного бита на две подстановки быстрее возрастает зависимость битов результата от битов исходных данных. Это называется **лавинным эффектом**. DES спроектирован так, чтобы как можно быстрее добиться зависимости каждого бита шифротекста от каждого бита открытого текста и каждого бита ключа.

Перестановка с расширением показана на 9-й. Иногда она называется **E-блоком** (от expansion). Для каждого 4-битового входного блока первый и четвертый бит представляют собой два бита выходного блока, а второй и третий биты - один бит выходного блока. В 7-й показано, какие позиции результата соответствуют каким позициям исходных данных. Например, бит входного блока в позиции 3 переместится в позицию 4 выходного блока, а бит входного блока в позиции 21 - в позиции 30 и 32 выходного блока.

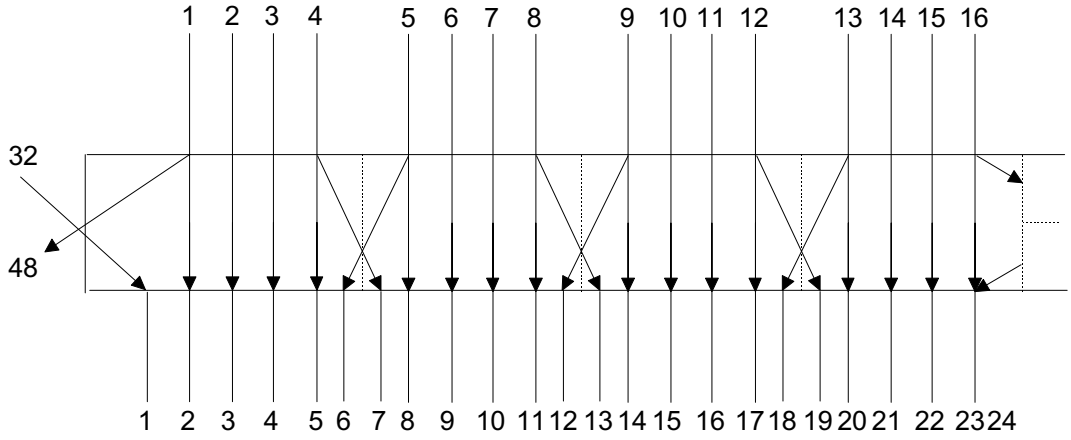


Рис. 12-3. Перестановка с расширением.

Хотя выходной блок больше входного, каждый входной блок генерирует уникальный выходной блок.

Табл. 12-5.
Перестановка с расширением

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,	21,	22,	23,	24,	25,
24,	25,	26,	27,	28,	29,	28,	29,	30,	31,	32,	1

Подстановка с помощью S-блоков

После объединения сжатого блока с расширенным блоком с помощью XOR над 48-битовым результатом выполняется операция подстановки. Подстановки производятся в восьми **блоках подстановки**, или **S-блоках** (от substitution). У каждого S-блока 6-битовый вход и 4-битовый выход, всего используется восемь различных S-блоков. (Для восьми S-блоков DES потребуется 256 байтов памяти.) 48 битов делятся на восемь 6-битовых подблока. Каждый отдельный подблок обрабатывается отдельным S-блоком: первый подблок - S-блоком 1, вт о-рой - S-блоком 2, и так далее. См. 8-й.

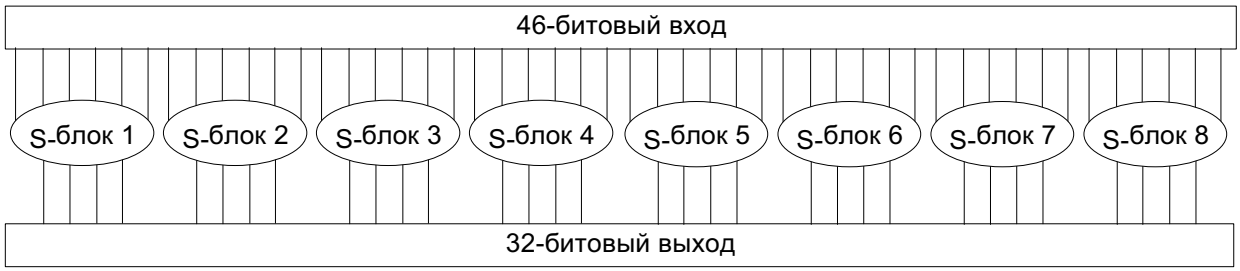


Рис. 12-4. Подстановка - S-блоки.

Каждый S-блок представляет собой таблицу из 2 строк и 16 столбцов. Каждый элемент в блоке является 4-битовым числом. По 6 входным битам S-блока определяется, под какими номерами столбцов и строк искать выходное значение. Все восемь S-блоков показаны в 6-й.

Табл. 12-6.
S-блоки

S-блок 1:															
14,	4,	13,	1,	2,	15,	11,	8,	3,	10,	6,	12.,	5,	9,	0,	7,
0,	15,	7,	4,	14,	2,	13,	1,	10,	6,	12.,	11,	9,	5,	3,	8,

4, 1, 14, 8, 13, 6, 2, 11, 15, 12, 9, 7, 3, 10, 5, 0,
15, 12, 8, 2, 4, 9, 1, 7, 5, 11, 3, 14, 10, 0, 6, 13,

S-блок 2:

15, 1, 8, 14, 6, 11, 3, 4, 9, 7, 2, 13, 12, 0, 5, 10,
3, 13, 4, 7, 15, 2, 8, 14, 12, 0, 1, 10, 6, 9, 11, 5,
0, 14, 7, 11, 10, 4, 13, 1, 5, 8, 12, 6, 9, 3, 2, 15,
13, 8, 10, 1, 3, 15, 4, 2, 11, 6, 7, 12, 0, 5, 14, 9,

S-блок 3:

10, 0, 9, 14, 6, 3, 15, 5, 1, 13, 12, 7, 11, 4, 2, 8,
13, 7, 0, 9, 3, 4, 6, 10, 2, 8, 5, 14, 12, 11, 15, 1,
13, 6, 4, 9, 8, 15, 3, 0, 11, 1, 2, 12, 5, 10, 14, 7,
1, 10, 13, 0, 6, 9, 8, 7, 4, 15, 14, 3, 11, 5, 2, 12,

S-блок 4:

7, 13, 14, 3, 0, 6, 9, 10, 1, 2, 8, 5, 11, 12, 4, 15,
13, 8, 11, 5, 6. 15, 0, 3, 4, 7, 2, 12, 1, 10, 14, 9,
10, 6, 9, 0, 12, 11, 7, 13, 15, 1, 3, 14, 5, 2, 8, 4,
3, 15, 0, 6, 10, 1, 13, 8, 9, 4, 5, 11, 12, 7, 2, 14,

S-блок 5:

2, 12, 4, 1, 7, 10, 11, 6, 8, 5, 3, 15, 13, 0, 14, 9,
14, 11, 2, 12, 4, 7, 13, 1, 5, 0, 15, 10, 3, 9, 8, 6,
4, 2, 1, 11, 10, 13, 7, 8, 15, 9, 12, 5, 6, 3, 0, 14,
11, 8, 12, 7, 1, 14, 2, 13, 6, 15, 0, 9, 10, 4, 5, 3,

S-блок 6:

12, 1, 10, 15, 9, 2, 6, 8, 0, 13, 3, 4, 14, 7, 5, 11,
10, 15, 4, 2, 7, 12, 9, 5, 6, 1, 13, 14, 0, 11, 3, 8,
9, 14, 15, 5, 2, 8, 12, 3, 7, 0, 4, 10, 1, 13, 11, 6,
4, 3, 2, 12, 9, 5, 15, 10, 11, 14, 1, 7, 6, 0, 8, 13,

S-блок 7:

4, 11, 2, 14, 15, 0, 8, 13, 3, 12, 9, 7, 5, 10, 6, 1,
13, 0, 11, 7, 4, 9, 1, 10, 14, 3, 5, 12, 2, 15, 8, 6,
1, 4, 11, 13, 12, 3, 7, 14, 10, 15, 6, 8, 0, 5, 9, 2,
6, 11, 13, 8, 1, 4, 10, 7, 9, 5, 0, 15, 14, 2, 3, 12,

S-блок 8:

13, 2, 8, 4, 6, 15, 11, 1, 10, 9, 3, 14, 5, 0, 12, 7,
1, 15, 13, 8, 10, 3, 7, 4, 12, 5, 6, 11, 0, 14, 9, 2,
7, 11, 4, 1, 9, 12, 14, 2, 0, 6, 10, 13, 15, 3, 5, 8,
2, 1, 14, 7, 4, 10, 8, 13, 15, 12, 9, 0, 3, 5, 6, 11

Входные биты особым образом определяют элемент S-блока. Рассмотрим 6-битовый вход S-блока: b_1, b_2, b_3, b_4, b_5 и b_6. Биты b_1 и b_6 объединяются, образуя 2-битовое число от 0 до 3, соответствующее строке таблицы. Средние 4 бита, с b_2 по b_5, объединяются, образуя 4-битовое число от 0 до 15, соответствующее столбцу табл ицы.

Например, пусть на вход шестого S-блока (т.е., биты функции XOR с 31 по 36) попадает 110011. Первый и последний бит, объединяясь, образуют 11, что соответствует строке 3 шестого S-блока. Средние 4 бита образуют 1001, что соответствует столбцу 9 того же S-блока. Элемент S-блока 6, находящийся на пересечении строки 3 и столбца 9, - это 14. (Не забывайте, что строки и столбцы нумеруются с 0, а не с 1.) Вместо 110011 подставляется 1110.

Конечно же, намного легче реализовать S-блоки программно в виде массивов с 64 элементами. Для этого потребуется переупорядочить элементы, что не является трудной задачей. (Изменить индексы, не изменяя порядок элементов, недостаточно. S-блоки спроектированы очень тщательно.) Однако такой способ описания S-блоков помогает понять, как они работают. Каждый S-блок можно рассматривать как функцию подстановки 4-битового элемента: b_2 по b_5являются входом, а некоторое 4-битовое число - результатом. Биты b_1 и b_6 определяются соседними блоками, они определяют одну из четырех функций подстановки, возможных в данном S-блоке.

Подстановка с помощью S-блоков является ключевым этапом DES. Другие действия алгоритма линейны и легко поддаются анализу. S-блоки нелинейны, и именно они в большей степени, чем все остальное, обеспечивают безопасность DES.

В результате этого этапа подстановки получаются восемь 4-битовых блоков, которые вновь объединяются в единый 32-битовый блок. Этот блок поступает на вход следующего этапа - перестановки с помощью P-блоков.

Перестановка с помощью P-блоков

32-битовый выход подстановки с помощью S-блоков, перетасовываются в соответствии с P-блоком. Эта перестановка перемещает каждый входной бит в другую позицию, ни один бит не используется дважды, и ни один бит не игнорируется. Этот процесс называется прямой перестановкой или просто перестановкой. Позиции, в которые перемещаются биты, показаны в 5-й. Например, бит 21 перемещается в позицию 4, а бит 4 - в позицию 31.

Табл. 12-7.
Перестановка с помощью P-блоков

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

Наконец, результат перестановки с помощью P-блока объединяется посредством XOR с левой половиной первоначального 64-битового блока. Затем левая и правая половины меняются местами, и начинается следующий этап.

Заключительная перестановка

Заключительная перестановка является обратной по отношению к начальной перестановки и описана в 4-й. Обратите внимание, что левая и правая половины не меняются местами после последнего этапа DES, вместо этого объединенный блок $R_{16}L_{16}$ используется как вход заключительной перестановки. В этом нет ничего особенного, перестановка половинок с последующим циклическим сдвигом привела бы к точно такому же результату. Это сделано для того, чтобы алгоритм можно было использовать как для шифрования, так и для дешифрования.

Табл. 12-8.
Заключительная перестановка

40,	8,	48,	16,	56,	24,	64,	32,	39,	7,	47,	15,	55,	23,	63,	31,
38,	6,	46,	14,	54,	22,	62,	30,	37,	5,	45,	13,	53,	21,	61,	29,
36,	4,	44,	12,	52,	20,	60,	28,	35,	3,	43,	11,	51,	19,	59,	27,
34,	2,	42,	10,	50,	18,	58,	26,	33,	1,	41,	9,	49,	17,	57,	25

Дешифрирование DES

После всех подстановок, перестановок, операций XOR и циклических сдвигов можно подумать, что алгоритм дешифрирования, резко отличаясь от алгоритма шифрования, точно также запутан. Напротив, различные компоненты DES были подобраны так, чтобы выполнялось очень полезное свойство: для шифрования и дешифрования

рирования используется один и тот же алгоритм.

DES позволяет использовать для шифрования или дешифрирования блока одну и ту же функцию. Единс т-венное отличие состоит в том, что ключи должны использоваться в обратном порядке. То есть, если на этапах шифрования использовались ключи K_1, K_2, K_3, ..., K_{16}, то ключами дешифрирования будут K_{16}, K_{15}, K_{14}, ..., K_1. Алгоритм, который создает ключ для каждого этапа, также цикличен. Ключ сдвигается направо, а число поз и-ций сдвига равно 0, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1.

Режимы DES

FIPS PUB 81 определяет четыре режима работы: ECB, CBC, OFB и CFB (см. главу 9) [1143]. Банковские стандарты ANSI определяют для шифрования ECB и CBC, а для проверки подлинности - CBC и n-битовый CFB [52].

В мире программного обеспечения сертификация обычно не важна. Из-за своей простоты в большинстве существующих коммерческих программ используется ECB, хотя этот режим наиболее чувствителен к вскр ытию. CBC используется редко несмотря на то, что он лишь незначительно сложнее, чем ECB, и обеспечивает большую безопасность.

Аппаратные и программные реализации DES

Об эффективных аппаратных и программных реализациях алгоритма много писалось [997, 81, 533, 534, 437, 738, 1573, 176, 271, 1572]. Утверждается, что самой быстрой является микросхема DES, разработанная в Digital Equipment Corporation [512]. Она поддерживает режимы ECB и CBC и основана на вентильной матрице GaAs, состоящей из 50000 транзисторов. Данные могут зашифровываться и дешифрироваться со скоростью 1 гигабит в секунду, обрабатывая 16.8 миллионов блоков в секунду. Это впечатляет. Параметры ряда коммерческих ми к-росхем DES приведены в 3-й. Кажущиеся противоречия между тактовой частотой и скоростью обработки да н-ных обусловлены конвейеризацией внутри микросхемы, в которой может быть реализовано несколько раб о-тающих параллельно DES-механизмов.

Наиболее выдающейся микросхемой DES является 6868 VLSI (ранее называвшаяся "Gatekeeper" - Вратарь). Она не только может выполнять шифрование DES за 8 тактов (лабораторные прототипы могут делать это за 4 такта), но также выполнять троекратный DES в режиме ECB за 25 тактов, а троекратный DES в режимах OFB или CBC - за 35 актов. Мне это кажется невозможным, но уверяю вас, она именно так и работает.

Программная реализация DES на мэйнфрейме IBM 3090 может выполнить 32000 шифрований DES в секу нду. На других платформах скорость ниже, но все равно достаточно велика. В 2-й [603, 793] приведены действ и-тельные результаты и оценки для различных ми кропроцессоров Intel и Motorola.

Табл. 12-9.
Коммерческие микросхемы DES

Производитель	Микросхема	Год	Тактовая частота	Скорость данных	Доступность
AMD	Am9518	1981	3 МГц	l.3 Мбайт/с	Н
AMD	Am9568	?	4 МГц	l.5 Мбайт/с	Н
AMD	AmZ8068	1982	4 МГц	l.7 Мбайт/с	Н
AT&T	T7000A	1985	?	l.9 Мбайт/с	Н
CE-Infosys	SuperCrypt CE99C003	1992	20 МГц	12.5 Мбайт/с	Д
CE-Infosys	SuperCrypt CE99C003A	1994	30 МГц	20.0 Мбайт/с	Д
Cryptech	Cry12C102	1989	20 МГц	2.8 Мбайт/с	Д
Newbridge	CA20C03A	1991	25 МГц	3.85 Мбайт/с	Д
Newbridge	CA20C03W	1992	8 МГц	0.64 Мбайт/с	Д
Newbridge	CA95C68/18/09	1993	33 МГц	14.67 Мбайт/с	Д
Pijnenburg	PCC100	?	?	2.5 Мбайт/с	Д
Semaphore Communications	Roadrunner284	?	40 МГц	35.5 Мбайт/с	Д

VLSI Technology	VM007	1993	32 МГц	200.0 Мбайт/с	Д
VLSI Technology	VM009	1993	33 МГц	14.0	Д
VLSI Technology	6868	1995	32 МГц	64.0 Мбайт/с	Д
Western Digital	WD2001/2002	1984	3 МГц	0.23 Мбайт/с	Н

Табл. 12-10.
Скорости DES на различных микропроцессорах и компьютерах

Процессор	Скорость (в МГц)	Блоки DES (в с)
8088	4.7	370
68000	7.6	900
80286	6	1100
68020	16	3500
68030	16	3900
80386	25	5000
68030	50	10000
68040	25	16000
68040	40	23000
80486	66	43000
Sun ELC		26000
HyperSparc		32000
RS6000-350		53000
Sparc 10/52		84000
DEC Alpha 4000/610		154000
HP9000/887	125	196,000

12.3 Безопасность DES

Люди давно интересуются безопасностью DES [458]. Было много рассуждений о длине ключа, количестве итераций и схеме S-блоков. S-блоки были наиболее таинственными - какие-то константы, без видимого объя снения для чего и зачем они нужны. Хотя IBM утверждала, что работа алгоритма была результатом 17 человеколет интенсивного криптоанализа, некоторые люди опасались, что NSA вставило в алгоритм лазейку, которая позволит агентству легко дешифрировать перехваченные соо бщения.

Комитет по разведке Сената США чрезвычайно тщательно расследовал этот вопрос в 1978 году. Результаты работы комитета были засекречены, но в открытых итогах этого расследования с NSA были сняты все обвин ения в неуместном вмешательстве в проектирование алгоритма [1552]. "Было сказано, что NSA убедило IBM в достаточности более короткого ключа, косвенно помогло разработать структуры S-блоков и подтвердило, что в окончательном варианте DES, с учетом всех знаний NSA, отсутствовали статистические или математические бреши " [435]. Однако, так как правительство не опубликовало подробности расследования, многих людей уб едить не удалось.

Тачмен (Tuchman) и Майер (Meyer), разработавшие DES криптографы IBM, заявили, что NSA не изменяло проект [841]:

> Их основным подходом был поиск сильных подстановок, перестановок и функций планирования ключей. . . . IBM по просьбе NSA засекретило информацию, касающуюся критериев выбора. ... "NSA сообщило нам, что мы самостоятельно зан о-во открыли ряд секретов, используемых для создания их собственных алгоритмов", - об ъясняет Тачмен.

Позже в одной из статей Тачмен писал: "Алгоритм DES был полностью разработан внутри IBM ее сотру дниками. NSA не продиктовало ни единой связи!" Тачмен подтвердил это утверждение в своем докладе по ист о-рии DES на Национальной конференции по компьютерной безопасности (National Computer Security Conference)

в 1992 году.

С другой стороны, Копперсмит писал [373, 374]: "Агентство национальной безопасности (NSA) также пом о-гало IBM техническими советами." А Конхейм (Konheim) утверждал: "Мы послали S-блоки в Вашингтон. Они вернулись полностью переработанными. Мы проверили их, и они прошли нашу проверку." На этот факт и сс ы-лаются как на доказательство, что NSA вставило лазейку в DES. По вопросу о каком-либо преднамеренном о с-лаблении DES NSA заявило [363]:

> Относительно Стандарта шифрования данных (DES) мы считаем, что ответ на ваш вопрос о роли NSA в разработке DES содержится в опубликованных итогах расследования Комитета Сената по разведке, проведенного в 1978 году. В сообщении Комитета указывается, что NSA никоим образом не искажало алгоритм, и что безопасность, предоставляемая DES для несе к-ретных данных, с целью защиты которых он и был разработан, была более чем адекватна в течение по крайней мере 5-10 лет. Короче говоря, NSA не вносило и не пыталось вносить никаких ослаблений в алгоритм DES.

Тогда почему они изменили S-блоки? Может быть, чтобы гарантировать, что лазейка не будет встроена в DES самой IBM. У NSA не было причин доверять исследователям IBM, и оно могло решить, что не до конца исполнит свой долг, если не обеспечит отсутствие лазеек в DES. Задание S-блоков и могло быть одним из сп о-собов гарантировать это.

Совсем недавно новые результаты криптоанализа прояснили этот вопрос, который в течение многих лет был предметом спекуляций.

Слабые ключи

Из-за того, что первоначальный ключ изменяется при получении подключа для каждого этапа алгоритма, определенные первоначальные ключи являются **слабыми** [721, 427]. Вспомните, первоначальное значение расщепляется на две половины, каждая из которых сдвигается независимо. Если все биты каждой половины равны 0 или 1, то для всех этапов алгоритма используется один и тот же ключ. Это может произойти, если ключ состоит из одних 1, из одних 0, или если одна половина ключа состоит из одних 1, а другая - из одних 0. Кроме того, у два слабых ключа обладают др угими свойствами, снижающими их безопасность [427].

Четыре слабых ключа показаны в шестнадцатиричном виде в 1-й. (Не забывайте, что каждый восьмой бит - это бит четности.)

Табл. 12-11.
Слабые ключи DES

Значение слабого ключа (с битами четн ости)				Действительный ключ
0101	0101	0101	0101	0000000 0000000
1F1F	1F1F	0E0E	0E0E	0000000 FFFFFFF
E0E0	E0E0	F1F1	F1F1	FFFFFFF 0000000
FEFE	FEFE	FEFE	FEFE	FFFFFFF FFFFFFF

Кроме того, некоторые пары ключей при шифровании переводят открытый текст в идентичный шифротекст. Иными словами, один из ключей пары может расшифровать сообщения, зашифрованные другим ключом пары. Это происходит из-за метода, используемого DES для генерации подключей - вместо 16 различных подключей эти ключи генерируют только два различных подключа. В алгоритме каждый из этих подключей используется восемь раз. Эти ключи, называемые **полуслабыми ключами**, в шестнадцатиричном виде приведены в 0-й.

Табл. 12-12.
Полуслабые пары ключей DES

01FE	01FE	01FE	01FE	и	FE01	FE01	FE01	FE01
1FE0	1FE0	0EF1	0EF1	и	E01F	E01F	F10E	F10E
01E0	01E0	01F1	01F1	и	E001	E001	F101	F101
1FFE	1EEE	0EFE	0EFE	и	FE1F	FE1F	FE0E	FE0E
011F	011F	010E	010E	и	1F01	1F01	0E01	0E01
E0FE	E0FE	F1FE	F1FE	и	FEE0	FEE0	FEE1	FEE1

Ряд ключей генерирует только четыре подключа, каждый из которых четыре раза используется в алгоритме.

Эти **возможно слабые ключи** перечислены в -1-й.

Табл. 12-13.
Возможно слабые ключи DES

1F	1F	01	01	0E	0E	01	01
01	1F	1F	01	01	0E	0E	01
1F	01	01	1F	0E	01	01	0E
01	01	1F	1F	01	01	0E	0E
E0	E0	01	01	F1	F1	01	01
FE	FE	01	01	FE	FE	01	01
FE	E0	1F	01	FE	F1	0E	01
E0	FE	1F	01	F1	FE	0E	01
FE	E0	01	1F	FE	F1	01	0E
E0	FE	01	1F	F1	FE	01	0E
E0	E0	1F	1F	F1	F1	0E	0E
FE	FE	1F	1F	FE	FE	0E	0E
FE	1F	E0	01	FE	0E	F1	01
E0	1F	FE	01	F1	0E	FE	01
FE	01	E0	1F	FE	01	F1	0E
E0	01	FE	1F	F1	01	FE	0E
01	E0	E0	01	01	F1	F1	01
1F	FE	E0	01	0E	FE	F0	01
1F	E0	FE	01	0E	F1	FE	01
01	FE	FE	01	01	FE	FE	01
1F	E0	E0	1F	0E	F1	F1	0E
01	FE	E0	1F	01	FE	F1	0E
01	E0	FE	1F	01	F1	FE	0E
1F	FE	FE	1F	0E	FE	FE	0E

E0	01	01	E0	F1	01	01	F1
FE	1F	01	E0	FE	0E	01	F1
FE	01	1F	E0	FE	01	0E	F1
E0	1F	1F	E0	F1	0E	0E	F1
FE	01	01	FE	FE	01	01	FE
E0	1F	01	FE	F1	0E	01	FE
E0	01	1F	FE	F1	01	0E	FE
FE	1F	1F	FE	FE	0E	0E	FE
1F	FE	01	E0	0E	FE	01	F1
01	FE	1F	E0	01	FE	0E	F1
1F	E0	01	FE	0E	F1	01	FE
01	E0	1F	FE	01	F1	0E	FE
01	01	E0	E0	01	01	F1	F1
1F	1F	E0	E0	0E	0E	F1	F1
1F	01	FE	E0	0E	01	FE	F1
01	1F	FE	E0	01	0E	FE	F1
1F	01	E0	FE	0E	01	F1	FE
01	1F	E0	FE	01	0E	F1	FE
01	01	FE	FE	01	01	FE	FE
1F	1F	FE	FE	0E	0E	FE	FE
FE	FE	E0	E0	FE	FE	F1	F1
E0	FE	FE	E0	F1	FE	FE	F1
FE	E0	E0	FE	FE	F1	F1	FE
E0	E0	FE	FE	F1	F1	FE	FE

Прежде, чем порицать DES слабые ключи, обратите внимание на то, что эти 64 ключа - это крошечная часть полного набора из 72057594037927936 возможных ключей. Если вы выбираете ключ случайно, вероятность выбрать один из слабых ключей пренебрежимо мала. Если вы настоящий параноик, можете всегда проверять "на слабость" сгенерированный ключ. Некоторые думают, что нечего и беспокоиться на этот счет. Другие у т-верждают, что проверка очень легка, почему бы ее и не в ыполнить.

Дальнейший анализ слабых и полуслабых ключей приведен в [1116]. Других слабых ключей в процессе и с-следований найдено не было.

Ключи-дополнения

Выполним побитное дополнение ключа, заменяя все 0 на 1 и все 1 - на 0. Теперь, если блок открытого текста зашифрован оригинальным ключом, то дополнение ключа при шифровании превратит дополнение блока о т-крытого текста в дополнение блока шифротекста. Если x' обозначает дополнение x, то следующее верно:

$E_K(P) = C$

$E_K(P') = C'$

В этом нет ничего таинственного. На каждом этапе после перестановки с расширением подключи подверг а-ются операции XOR с правой половиной. Прямым следствием этого факта и является приведенное свойство комплиментарности.

Это означает, что при выполнении вскрытия DES с выбранным открытым текстом нужно проверять только половину возможных ключей: 2^{55} вместо 2^{56} [1080]. Эли Бихам (Eli Biham) и Ади Шамир показали [172], что существует вскрытие с известным открытым текстом, имеющее ту же сложность, для которого нужно не меньше 2^{33} известных открытых текстов.

Остается вопросом, является ли такое свойство слабостью, так как в большинстве сообщений нет компл и-ментарных блоков открытого текста (для случайного открытого текста шансы "против" чрезвычайно велики), а пользователей можно предупредить не пользоваться дополняющими.

Алгебраическая структура

Все возможные 64-битовые блоки открытого текста можно отобразить на 64-битовые блоки шифротекста

2^{64}! Различными способами. Алгоритм DES, используя 56-битовый ключ, предоставляет нам 2^{56} (приблизительно 10^{17}) таких отображений. Использование многократного шифрования на первый взгляд позв оляет значительно увеличить долю возможных отображений. Но это правильно только, если действие DES не обладает определенной алгебраической структурой.

Если бы DES был **замкнутым**, то для любых K_1 и K_2 всегда существовало бы такое K_3, что

$$E_{K_2}(E_{K_1}(P)) = E_{K_3}(P)$$

Другими словами, операция шифрования DES образовала бы группу, и шифрование набора блоков открыт ого текста последовательно с помощью K1 и K2 было бы идентично шифрованию блоков ключом K3. Что еще хуже, DES был бы чувствителен к вскрытию "встреча посередине" с известным открытым текстом, для которого потребовалось бы только 2^{28} этапов [807].

Если бы DES был **чистым**, то для любых K_1, K_2 и K_3 всегда существовало бы такое K_4, что

$$E_{K_3}(E_{K_2}(E_{K_1}(P))) = E_{K_4}(P)$$

Тройное шифрование было бы бесполезным. (Заметьте, что замкнутый шифр обязательно является и чи стым, но чистый шифр не обязательно является замкнутым.)

Ряд подсказок можно найти в ранней теоретической работе Дона Копперсмита, но этого недостаточно [377]. Различные криптографы пытались решить эту проблему [588, 427, 431, 527, 723, 789]. В повторяющихся эксп ериментах собирались "неопровержимые доказательства" того, что DES не является группой [807, 371, 808, 1116, 809], но только в 1992 году криптографам удалось это доказать окончательно [293]. Копперсмит утве рждает, что команда IBM знала об этом с самого н ачала.

Длина ключа

В оригинальной заявке фирмы IBM в NBS предполагалось использовать 112-битовый ключ. К тому времени, когда DES стандартом, длина ключа уменьшилась до 56 бит. Многие криптографы настаивали на более дли нном ключе. Основным их аргументом б ыло вскрытие грубой силой (см. раздел 7.1).

В 1976 и 1977 гг. Диффи и Хеллман утверждали, что специализированный параллельный компьютер для вскрытия DES, стоящий 20 миллионов долларов, сможет раскрыть ключ за день. В 1981 году Диффи увеличил время поиска до двух дней, а стоимость - до 50 миллионов долларов [491]. Диффи и Хеллман утверждали, что вскрытие в тот момент времени находилось за пределами возможностей любой организации, кроме подобных NSA, но что к 1990 году DES должен полностью утратить свою безопасность [714].

Хеллман [716] продемонстрировал еще один аргумент против малого размера ключа: разменивая объем п амяти на время, можно ускорить процесс поиска. Он предложил вычислять и хранить 2^{56} возможных результатов шифрования каждым возможным ключом единственного блока открытого текста. Тогда для взлома неизвестн ого ключа криптоаналитику потребуется только вставить блок открытого текста в шифруемый поток, вскрыть получившийся результат и найти ключ. Хеллман оценил стоимость такого устройства вскрытия в 5 миллионов долларов.

Аргументы за и против существования в каком-нибудь тайном бункере правительственного устройства вскрытия DES продолжают появляться. Многие указывают на то, что среднее время наработки на отказ для микросхем DES никогда не было большим настолько, чтобы обеспечивать работу устройства. В [1278] было показано, что этого возражения более чем достаточно. Другие исследователи предлагают способы еще больше ускорить процесс и уменьшить эффект отказа микросхем.

Между тем, аппаратные реализации DES постепенно приблизились к реализации требования о миллионе шифрований в секунду, предъявляемого специализированной машиной Диффи и Хеллмана. В 1984 году были выпущены микросхемы DES, способные выполнять 256000 шифрования в секунду [533, 534]. К 1987 году были разработаны микросхемы DES, выполняющие 512000 шифрований в секунду, и стало возможным появление варианта, способного проверять свыше миллиона ключей в секунду [738, 1573]. А в 1993 Майкл Винер (Michael Wiener) спроектировал машину стоимостью 1 миллион долларов, которая может выполнить вскрытие DES гр убой силой в среднем за 3.5 часа (см. раздел 7.1).

Никто открыто не заявил о создании этой машины, хоте разумно предположить, что кому-то это удалось. Миллион долларов - это не слишком большие деньги для большой и даже не очень большой страны.

В 1990 году два израильских математика, Бихам (Biham) и Шамир, открыли **дифференциальный крип тоанализ**, метод, который позволил оставить в покое вопрос длины ключа. Прежде, чем мы рассмотрим этот метод, вернемся к некоторым другим крит ическим замечаниям в адрес DES.

Количество этапов

Почему 16 этапов? Почему не 32? После пяти этапов каждый бит шифротекста является функцией всех б итов открытого текста и всех битов ключа [1078, 1080], а после восьми этапов шифротекст по сути представляет собой случайную функцию всех битов открытого текста и всех битов ключа [880]. (Это называется лавинным эффектом.) Так почему не остановиться после восьми этапов?

В течение многих лет версии DES с уменьшенным числом этапов успешно вскрывались. DES с тремя и ч етырьмя этапами был легко взломан в 1982 году [49]. DES с шестью этапами пал несколькими годами позже [336]. Дифференциальный криптоанализ Бихама и Шамира объяснил и это: DES с любым количеством этапов, меньшим 16, может быть взломан с помощью вскрытия с известным открытым текстом быстрее, чем с пом ощью вскрытия грубой силой. Конечно грубый взлом является более вероятным способом вскрытия, но интер есен тот факт, что алгоритм содержит ровно 16 этапов.

Проектирование S-блоков

Помимо уменьшения длины ключа NSA также обвиняют в изменении содержания S-блоков. Настаивая на подтверждении схемы S-блоков, NSA заявило, что детали алгоритма являются "чувствительными" и не могут быть опубликованы. Многие криптографы подозревали, что разработанные в NSA S-блоки содержат лазейку, позволяющую NSA легко выполнять криптоанализ алгоритма.

С момента появления алгоритма для анализа схемы и работы S-блоков были предприняты значительные усилия. В середине 70-х Lexar Corporation [961, 721] и Bell Laboratories [1120] исследовали работу S-блоков. Ни одно из исследований не обнаружило никаких слабостей, хотя оба исследования обнаружили непонятный сво йства. S-блоки имеют больше свойств, общих с линейным преобразованием, чем можно было ожидать при их формировании случайным образом. Команда Bell Laboratories констатировала, что S-блоки могут содержать скрытые лазейки, а доклад Lexar з авершался следующей фразой:

> В DES были найдены структуры, несомненно вставленные для повышения устойчивости системы к определенным типам вскрытия. Также были найдены структуры, кот орые, по видимому, ослабили систему.

С другой стороны этот доклад также содержал следующее предупреждение:

> ... проблема [поиска структур в S-блоках] усложняется из-за способности человеческого сознания находить в случайных данных структуры, которые в действител ьности вовсе не являются структурами.

На втором симпозиуме по DES Агентство национальной безопасности раскрыло ряд критериев проектиров ания S-блоков [229]. Но это не смогло снять всех подозрений, и спор продолжился [228, 422, 714, 1506, 1551].

В литературе про S-блоки писались удивительные вещи. Последние три бита результата четвертого S-блока могут быть получены тем же способом, что и первые, при помощи дополнения некоторых из входных битов [436, 438]. Различные, но тщательно подобранные входные данные для S-блоков могут давать одинаковый р езультат [436]. Можно получить результат одного этапа DES, меняя биты только в трех соседних S-блоках [487]. Шамир заметил, что элементы S-блоков, казалось, были несколько неустойчивы, но не собирался использовать эту неустойчивость для вскрытия [1423]. (Он упомянул об особенности пятого S-блока, но только спустя восемь лет линейный криптоанализ воспользовался этой особенностью.) Другие исследователи показали, что для пол учения S-блоков с наблюдаемыми характеристиками могли использоваться общеизвестные принципы проект ирования [266).

Дополнительные результаты

Были предприняты и другие попытки криптоанализировать DES. Один из криптографов искал закономерн ости, используя спектральные тесты [559]. Другие анализировали последовательность линейных множителей, но их вскрытие потерпело неудачу после восьми этапов [1297, 336, 531]. Неопубликованное вскрытие, выполне нное в 1987 году Дональдом Дэвисом (Donald Davies), использовало способ, с помощью которого перестановка с расширением повторяет биты в соседних S-блоках, это вскрытие также оказалось бесполезным после восьми этапов [172, 429].

12.4 Дифференциальный и линейный криптоанализ

Дифференциальный криптоанализ

В 1990 году Эли Бихам и Ади Шамир ввели понятие **дифференциального криптоанализа** [167, 168, 171, 172]. Это был новый, ранее неизвестный метод криптоанализа. Используя этот метод, Бихам и Шамир нашли способ вскрытия DES с использованием выбранного открытого текста, который был эффективнее вскрытия гр убой силой.

Дифференциальный криптоанализ работает с **парами шифротекстов**, открытые тексты которых содержат определенные отличия. Метод анализирует эволюцию этих отличий в процессе прохождения открытых текстов

через этапы DES при шифровании одним и тем же ключом.

Просто выберем пару открытых текстов с фиксированным различием. Можно выбрать два открытых текста случайным образом, лишь бы они отличались друг от друга определенным образом, криптоаналитику даже не нужно знать их значений. (Для DES термин "различие" определяется с помощью XOR. Для других алгоритмов этот термин может определяться по другому.) Затем, используя различия в получившихся шифротекстах, пр исвоим различные вероятности различным ключам. В процессе дальнейшего анализа следующих пар шифроте кстов один из ключей станет наиболее вероятным. Это и есть правильный ключ.

Подробности гораздо сложнее. На 7-й представлена функция одного этапа DES. Представьте себе пару вх одов, X и X', с различием ΔX. Выходы, Y и Y' известны, следовательно, известно и различие между ними ΔY. Известны и перестановка с расширением, и P-блок, поэтому известны ΔA и ΔC. B и B' неизвестны, но их разность ΔB известна и равна ΔA. (При рассмотрении различия XOR K_i с A и A' нейтрализуются.) Пока все просто. Фокус вот в чем: для любого заданного ΔA не все значения ΔC равновероятны. Комбинация ΔA и ΔC позволяет предположить значения битов для A XOR K_i и A' XOR K_i. Так как A и A' известны, это дает нам информацию о K_i.

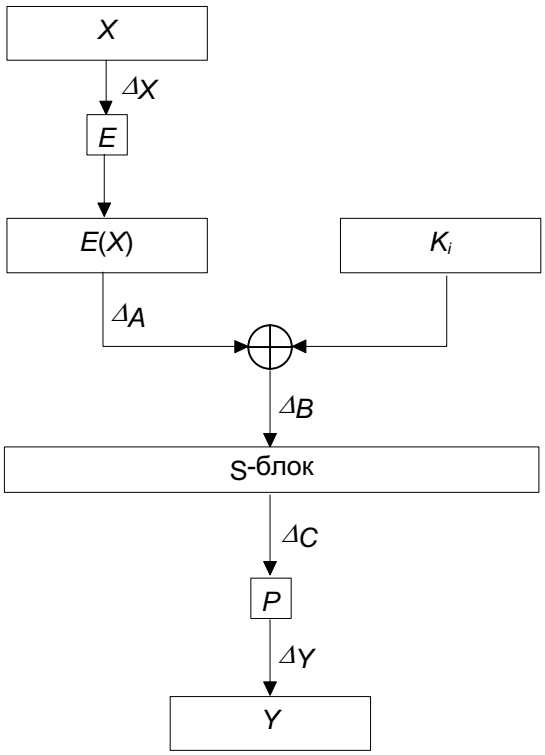


Рис. 12-5. Функция этапа DES.

Взглянем на последний этап DES. (При дифференциальном криптоанализе начальная и заключительная п ерестановки игнорируются. Они не влияют на вскрытие, только затрудняя объяснение.) Если мы сможем опред елить K_{16}, то мы получим 48 битов ключа. (Не забывайте, на каждом этапе подключ состоит из 48 битов 56-битового ключа.) Оставшиеся 8 битов мы можем получить грубым взломом. K_{16} даст нам дифференциальный криптоанализ.

Определенные различия пар открытых текстов обладают высокой вероятностью вызвать определенные ра зличия получаемых шифротекстов. Эти различия называются **характеристиками**. Характеристики распростр аняются на определенное количество этапов и по существу определяют прохождение этих этапов. Существуют входное различие, различие на каждом этапе и выходное различие - с определенной вероятностью.

Эти характеристики можно найти, создав таблицу, строки которой представляют возможные входы XOR (XOR двух различных наборов входных битов), столбцы - возможные результаты XOR, а элементы - сколько раз конкретный результат XOR встречается для заданного входа XOR. Такую таблицу можно сгенерировать для каждого из восьми S-блоков DES.

Например, на 6-йа показана характеристика одного этапа. Входное различие слева равно L, оно может быть произвольным. Входное различие справа равно 0. (У двух входов одинаковая правая половина, поэтому их ра зличие - 0.) Так как на входе функции этапа нет никаких различий, то нет различий и на выходе функции этапа. Следовательно, выходное различие левой части - $L \oplus 0 = L$, а выходное различие правой части - 0. Это трив иальная характеристика, она истинна с вероятностью 1.

На 6-йб показана менее очевидная характеристика. Снова, различие L левых частей произвольно. Входное различие правых частей равно 0x60000000, два входа отличаются только первым и третьим битами. С вероя т-ностью 14/64 различие на выходе функции этапа равно $L \oplus$ 0x00808200. Это означает, что выходное различие левых половин равно $L \oplus$ 0x00808200, а выходное различие правых половин - 0x60000000 (с вероятностью 14/64)

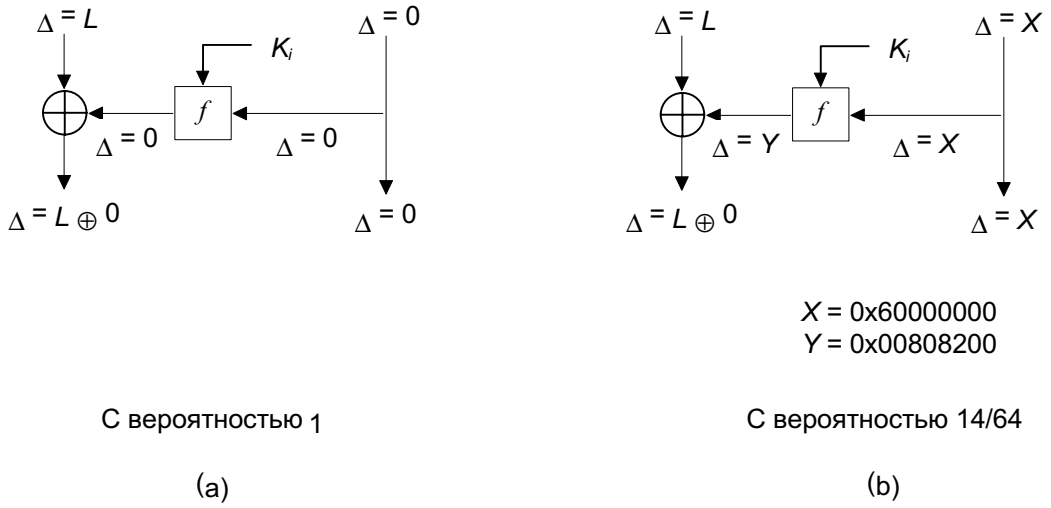


Рис. 12-6. Характеристики DES.

Различные характеристики можно объединять. Также, при условии, что этапы независимы, вероятности м о-гут перемножаться. На 5-й объединяются две ранее описанных характеристики. Входное различие слева равно 0x00808200, а справа - 0x60000000. В конце первого этапа входное различие и результат функции этапа нейтр а-лизуют друг друга, и выходное различие равно 0. Это различие поступает на вход второго этапа, окончательное выходное различие слева равно 0x60000000, а справа - 0. Вероятность этой двухэтапной характерист ики - 14/64.

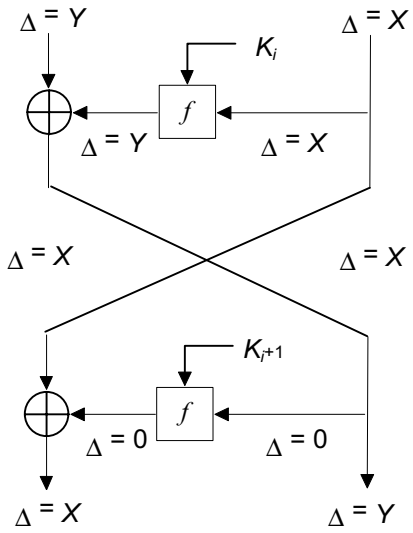


Рис. 12-7. Двухэтапная характеристика DES.

Пара открытых текстов, соответствующих характеристике, называется правильной парой, а пара открытых текстов, несоответствующих характеристике - неправильной парой. Правильная пара подсказывает правильный ключ этапа (для последнего этапа характеристики), неправильная пара - случайный ключ этапа.

Чтобы найти правильный ключ этапа, нужно просто собрать достаточное количество предположений. Один

из подключей будет встречаться чаще, чем все остальные. Фактически, правильный подключ возникнет из всех случайный возможных подключей.

Итак, дифференциальное основное вскрытие n-этапного DES дает 48-битовый подключ, используемый на этапе n, а оставшиеся 8 битов ключа получаются с помощью грубого взлома.

Но ряд заметных проблем все же остается. Во первых, пока вы не перейдете через некоторое пороговое зн ачение, вероятность успеха пренебрежимо мала. То есть, пока не будет накоплено достаточное количество да нных, выделить правильный подключ из шума невозможно. Кроме того, такое вскрытие не практично. Для хр анения вероятностей 2^{48} возможных ключей необходимо использовать счетчики, и к тому же для вскрытия п отребуется слишком много данных.

Бихам и Шамир предложили свой способ вскрытия. Вместо использования 15-этапной характеристики 16-этапного DES, они использовали 13-этапную характеристику и ряд приемов для получения последних нескол ьких этапов. Более короткая характеристика с большей вероятностью будет работать лучше. Они также испол ьзовали некоторые сложные математические приемы для получения вероятных 56-битовых ключей, которые и проверялись немедленно, таким образом устранялась потребность в счетчиках. Такое вскрытие достигает усп еха, как только находится правильная пара. Это позволяет избежать порогового эффекта и получить линейную зависимость для вероятности успеха. Если у вас в 1000 раз меньше пар, то вероятность успеха в 1000 раз мен ьше. Это звучит ужасно, но это намного лучше, чем порог. Всегда есть некоторая вероятность немедленной уд ачи.

Результаты являются весьма интересными. В -2-й проведен обзор лучших дифференциальных вскрытий DES с различным количеством этапов [172]. Первый столбец содержит количество этапов. Элементы следующих двух столбца представляют собой количество выбранных или известных открытых текстов, которые должны быть проверены для вскрытия, а четвертый столбец содержит количество действительно проанализированных открытых текстов. В последнем столбце приведена сложность анализа, после обнаружения требуемой пары.

Табл. 12-14. Вскрытие с помощью дифференциального криптоанализа

Количество этапов	Выбранные открытые тексты	Известные открытые тексты	Проанализированные открытые тексты	Сложность анализа
8	2^{14}	2^{38}	4	29
9	2^{24}	2^{44}	2	2^{32}†
10	2^{24}	2^{43}	2^{14}	2^{15}
11	2^{31}	2^{47}	2	2^{32}†
12	2^{31}	2^{47}	2^{21}	2^{21}
13	2^{39}	2^{52}	2	2^{32}†
14	2^{39}	2^{51}	2^{29}	2^{29}
15	2^{47}	2^{56}	27	2^{37}
16	2^{47}	2^{55}	2^{36}	2^{37}

† Сложность анализа для этих вариантов может быть значительно уменьшена за счет использования примерно в четыре раза большего количество открытых текстов и метода группировок.

Наилучшее вскрытие полного 16-этапного DES требует 2^{47} выбранных открытых текстов. Можно преобраз овать его к вскрытию с известным открытым текстом, но для него потребуется уже 2^{55} известных открытых тек стов. При анализе потребуется 2^{37} операций DES.

Дифференциальный криптоанализ эффективен против DES и аналогичных алгоритмов с постоянными S-блоками. Эффективность вскрытие сильно зависит от структуры S-блоков, блоки DES по счастливой случайн ости были оптимизированы против дифференциального криптоанализа. Для всех режимов работы DES - ECB, CBC, CFB и OFB - вскрытие с дифференциальным криптоанализом имеет одинаковую сложность [172].

Устойчивость DES может быть повышена путем увеличения количества этапов. Дифференциальный кри птоанализ с выбранным открытым текстом для DES с 17 или 18 этапами потребует столько же времени, сколько нужно для вскрытия грубой силой [160]. При 19 и более этапах дифференциальный криптоанализ становится невозможным, так как для него потребуется более, чем 2^{64} выбранных открытых текстов - не забудьте, DES и с пользует блоки размером 64 битов, поэтому для него существует только 2^{64} возможных открытых текстов. (В общем случае, вы можете доказать устойчивость алгоритма к дифференциальному криптоанализу, показав, что количество открытых текстов, необходимых для выполнения вскрытия, превышает количество возможных о ткрытых текстов.)

Нужно отметить ряд важных моментов. Во первых, это вскрытие в значительной степени теоретическое. О громные требования к времени и объему данных, необходимых для выполнения вскрытия с помощью дифф еренциального криптоанализа, находятся почти для всех вне пределов досягаемости. Чтобы получить нужные данные для выполнения такого вскрытия полного DES, вам придется почти три года шифровать поток выбра нных шифротекстов 1.5 Мегабит/с. Во вторых, это в первую очередь вскрытие с выбранным открытым текстом. Оно может быть преобразовано к вскрытию с известным открытым текстом, но вам придется просмотреть все пары "открытый текст/шифротекст" в поисках полезных. В случае полного 16-этапного DES это делает вскр ытие чуть менее эффективным по сравнению с грубой силой (вскрытие дифференциальным криптоанализом тр ебует $2^{55.1}$ операций, а вскрытие грубой силой - 2^{55}). Таким образом, правильно реализованный DES сохраняет устойчивость к дифференциальному криптоанализу.

Почему DES так устойчив к дифференциальному криптоанализу? Почему S-блоки оптимизированы так, что усложняют такое вскрытие насколько возможно? Почему используется ровно столько, а не больше этапов? П отому что создатели DES знали о дифференциальном анализе. Дон Копперсмит из IBM недавно писал [373, 374]:

> При проектировании использовались преимущества определенных криптоаналитических методов, особенно метода "дифференциального криптоанализа", который не был опубликован в открытой литературе. После дискуссий с NSA было р ешено, что раскрытие процесса проектирования раскроет и метод дифференциального криптоанализа, мощь которого может быть использована против многих шифров. Это, в свою очередь, сократило бы преимущество Соединенных Штатов перед другими странами в области криптографии.

Ади Шамир откликнулся, предложив Копперсмиту признаться, что с тех пор ему не удалось найти эффе ктивного способа вскрытия DES. Копперсмит предпочел отмолчат ься [1426].

Криптоанализ со связанными ключами

В 9-й показано количество битов, на которые циклически смещается ключ DES на каждом этапе: на 2 бита на каждом этапе, кроме этапов 1, 2, 9 и 16, когда ключ сдвигается на 1 бит. Почему?

Криптоанализ со связанными ключами похож на дифференциальный криптоанализ, но он изучает разл ичие между ключами. Вскрытие отличается от любого из ранее рассмотренных: криптоаналитик выбирает связь между парой ключей, но сами ключи остаются ему неизвестны. Данные шифруются обоими ключами. В варианте с известным открытым текстом криптоаналитику известны открытый текст и шифротекст данных, шифр ованных двумя ключами. В варианте с выбранным открытым текстом криптоаналитик пытается выбрать откр ытый текст, зашифрованный двумя ключами.

Модифицированный DES, в котором ключ сдвигается на два бита после каждого этапа, менее безопасен. Криптоанализ со связанными ключами может взломать такой вариант алгоритма, использовав только 2^{17} выбранных открытых текстов для выбранных ключей или 2^{33} известных открытых текстов для выбранных ключей [158, 163].

Такое вскрытие также не реализуемо на практике, но оно интересно по трем причинам. Во первых, это пе рвая попытка криптоаналитического вскрытия алгоритма генерации подключей в DES. Во вторых, это вскрытие не зависит от количества этапов криптографического алгоритма, он одинаково эффективен против DES с 16, 32 или 1000 этапами. И в третьих, DES невосприимчив к такому вскрытию. Изменение количества битов циклич еского сдвига мешает криптоанализу со связанными ключами.

Линейный криптоанализ

Линейный криптоанализ представляет собой другой тип криптоаналитического вскрытия, изобретенный Мицуру Мацуи (Mitsuru Matsui) [1016, 1015, 1017]. Это вскрытие использует линейные приближения для оп исания работы блочного шифра (в данном случае DES.)

Это означает, что если вы выполните операцию XOR над некоторыми битами открытого текста, затем над некоторыми битами шифротекста, а затем над результатами, вы получите бит, который представляет собой XOR некоторых битов ключа. Это называется линейным приближением, которое может быть верным с некот орой вероятностью p. Если $p \neq 1/2$, то это смещение можно использовать. Используйте собранные открытые те ксты и связанные шифротексты для предположения о значениях битов ключа. Чем больше у вас данных, тем вернее предположение. Чем больше смещение, тем б ыстрее вскрытие увенчается успехом.

Как определить хорошее линейное приближение для DES? Найдите хорошие одноэтапные линейные пр иближения и объедините их. (Начальная и заключительная перестановки снова игнорируются, так как они не влияют на вскрытие.) Взгляните на S-блоки. У них 6 входных битов и 4 выходных. Входные биты можно объ единить с помощью операции XOR 63 способами ($2^6 - 1$), а выходные биты - 15 способами. Теперь для каждого S-блока можно оценить вероятность того, что для случайно выбранного входа входная комбинация XOR равна некоторой выходной комбинации XOR. Если существует комбинация с достаточно большим смещением, то л инейный криптоанализ может сработать.

Если линейные приближения не смещены, то они будут выполняться для 32 из 64 возможных входов. Я и з-

бавлю вас от длительного изучения таблиц, наиболее смещенным S-блоком является пятый S-блок. Действ ительно, для 12 входов второй входной бит равен XOR всех четырех выходных битов. Это соответствует вероя тности 3/16 или смещению 5/16, что является самым большим смещением для всех S-блоков. (Шамир писал об этом в [1423], но не смог найти способа использовать.)

На 4-й показано, как воспользоваться этим для вскрытия функции этапа DES. b26 - это входной бит S-блока 5. (Я нумерую биты слева направо от 1 до 64. Мацуи игнорирует это принятое для DES соглашение и нумерует свои биты справа налево и от 0 до 63. Этого хватит, чтобы свести вас с ума.) c_{17}, c_{18}, c_{19}, c_{20} - это 4 выходных бита S-блока 5. Мы можем проследить b_{26} в обратном направлении от входа в S-блок. Для получения b_{26} бит объединяется с помощью XOR с битом подключа $K_{i,26}$. А бит X_{17} проходит через подстановку с расширением, чтобы превратиться в a_{26}. После S-блока 4 выходных бита проходят через P-блок, превращаясь в четыре выхо дных бита функции этапа: Y_3, Y_8, Y_{14} и Y_{25}. Это означает, что с вероятностью 1/2 - 5/6:

$X_{17} \oplus Y_3 \oplus Y_8 \oplus Y_{14} \oplus Y_{25} = K_{i,26}$

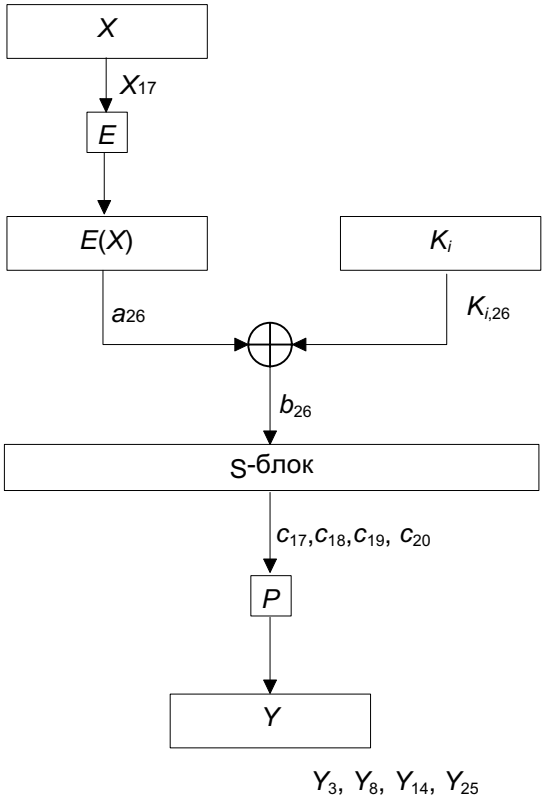
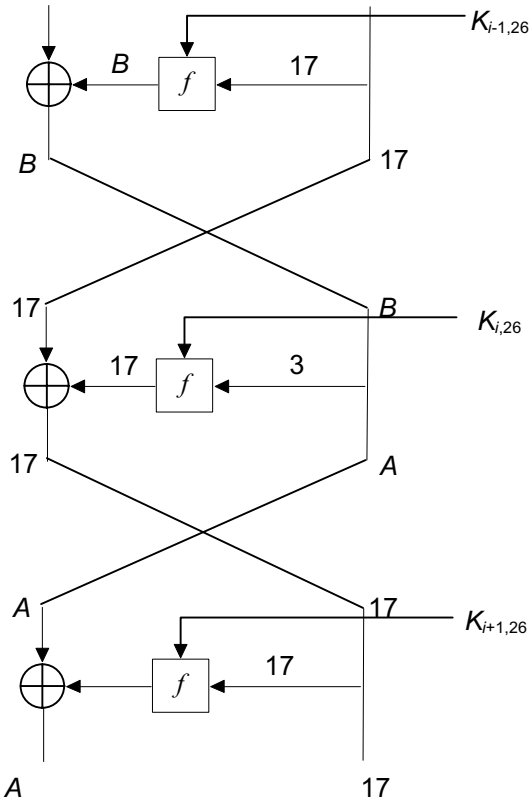


Рис. 12-8. 1-этапное линейное приближение для DES.

Способ, которым можно объединить линейные приближения для различных этапов, похож на тот, который обсуждался для дифференциального криптоанализа. На 3-й показано 3-этапное линейное приближение с вер оятностью 1/2+0.0061. Качество отдельных приближений различно: последнее очень хорошо, первое достаточно хорошо, а среднее - плохо. Но вместе эти три 1-этапных приближения дают очень хорошее трехэтапное пр иближение.



A=[3, 8, 14, 25] B=[8, 14, 25]

С вероятностью $1/2+6.1*10^{-3}$

Рис. 12-9. 3-этапное линейное приближение DES.

Базовое вскрытие должно использовать наилучшее линейное приближение для 16-этапного DES. Для него требуется 2^{47} известных открытых блоков, а результатом вскрытия является 1 бит ключа. Это не очень полезно. Если вы поменяете местами открытый текст и шифротекст и используете дешифрирование вместе с шифров анием, вы сможете получить 2 бита. Это все еще не очень полезно.

Существует ряд тонкостей. Используйте 14-этапное линейное приближение для этапов с 2 по 15. Попробуем угадать 6 битов подключа для S-блока 5 первого и последнего этапов (всего, таким образом, 12 битов ключа). Для эффективности выполняем линейный криптоанализ параллельно 2^{12} раз и выбираем правильный вариант, основываясь на вероятностях. Это раскрывает 12 битов и b_{26}, а поменяв местами открытый текст и шифротекст мы получим еще 13 битов. Для получения оставшихся 30 битов используйте исчерпывающий поиск. Существ уют и друге приемы, но описанный является основным.

При вскрытии таким образом полного 16 этапного DES ключ будет раскрыт в среднем с помощью 2^{43} известных открытых текстов. Программная реализации этого вскрытия, работая на 12 рабочих станциях HP9735, раскрыла ключ DES за 50 дней [1019]. В момент написания этой книги это наиболее эффективный способ вскрытия DES.

Линейный криптоанализ сильно зависит от структуры S-блоков, оказалось, что S-блоки DES не оптимизир ованы против такого способа вскрытия. Действительно, смещение в S-блоках, выбранных для DES, находится между 9 и 16 процентами, что не обеспечивает надежной защиты против линейного криптоанализа [1018]. С огласно Дону Копперсмиту [373, 374] устойчивость к линейному криптоанализу "не входило в число критериев проектирования DES". Либо разработчикам не было известно о линейном криптоанализе, либо при проектир овании они отдали преимущество устойчив ости против известного им еще более мощного средства вскрытия.

Линейный криптоанализ новее, чем дифференциальный, и в ближайшее время возможно дальнейшее пр одвижение в этом направлении. Некоторые идеи выдвинуты в [1270, 811], но не ясно, можно ли их эффективно применить против полного DES. Однако они очень хорошо работают против вариантов с уменьшенным числом этапов.

Дальнейшие направления

Был предпринят ряд попыток расширить концепцию дифференциального криптоанализа на дифференциалы более высоких порядков [702, 161, 927, 858, 860]. Ларс Кнудсен (Lars Knudsen) использует нечто, называемое частичными дифференциалами для вскрытия 6-этапного DES. Этот метод требует 32 выбранных открытых текста и 20000 шифрований [860]. Но этот метод слишком нов, чтобы можно было утверждать, что он облегчит вскрытие полного 16-этапного DES.

Другим способом вскрытия является дифференциально-линейный криптоанализ - объединение дифференциального и линейного криптоанализа. Сьюзен Лангфорд (Susan Langford) и Хеллман предлагают вскрытие 8-этапного DES, которое раскрывает 10 битов ключа с вероятностью успеха 80 процентов, используя 512 выбранных открытых текстов, и с вероятностью успеха 95 процентов, используя 768 выбранных открытых текстов [938]. После вскрытия необходим поиск грубой силой в оставшемся пространстве ключей (2^{46} возможных ключей). Хотя по времени это вскрытие сравнимо с предыдущими способами, для него требуется намного меньше открытых текстов. Однако расширение этого метода на большее количество этапов легким не кажется.

Но этот метод нов, и работа продолжается. В ближайшие годы возможны заметные успехи. Может быть у успеха добьется сочетание этого вскрытия с дифференциальным криптоанализом более высоких порядков. Кто знает?

12.5 Реальные критерии проектирования

После появления публикаций о дифференциальном криптоанализе IBM раскрыла критерии проектирования S-блоков и P-блока [373, 374]. Критериями проектирования S-блоков являлись:

— У каждого S-блока 6 входных битов и 4 выходных бита. (Это самый большой размер, который мог быть реализован в одной микросхеме по технологии 1974 года.)

— Ни один выходной бит S-блока не должен быть слишком близок к линейной функции входных битов.

— Если зафиксировать крайние левый и правый биты S-блока, изменяя 4 средних бита, то каждый возможный 4-битовый результат получается только один раз.

— Если два входа S-блока отличаются только одним битом, результаты должны отличаться по крайней мере на 2 бита.

— Если два входа S-блока отличаются только двумя центральными битами, результаты должны отличаться по крайней мере на 2 бита.

— Если два входа S-блока отличаются двумя первыми битами, а последние их последние 2 бита совпадают, результаты не должны быть одинаковыми.

— Для любого ненулевого 6-битового отличия между входами, не более, чем 8 из 32 пар входов могут приводить на выходе к одинаковому различию.

— Аналогичный предыдущему критерий, но для случая трех активных S-блоков.

Критериями проектирования P-блока являлись:

— 4 выходных бита каждого S-блока на этапе i распределены так, чтобы 2 из них влияют на средние биты S-блоков на этапе i + 1, а другие 2 бита влияют на последние биты.

— 4 выходных бита каждого S-блока влияют на шесть различных S-блоков, никакие 2 не влияют на один и тот же S-блок.

— Если выходной бит одного S-блока влияет на средние биты другого S-блока, то выходной бит этого другого S-блока не может влиять на средние биты первого S-блока.

Эта работа продолжала обсуждение критериев. Сегодня совсем нетрудно генерировать S-блоки, но в начале 70-х это было нелегкой задачей. Тачмен говорил, что программы, готовившие S-блоки, работали месяцами.

12.6 Варианты DES

Многократный DES

В ряде реализаций DES используется трехкратный DES (см. 2-й) [55]. Так как DES е является группой, полученный шифротекст гораздо сложнее вскрыть, используя исчерпывающий поиск: 2^{112} попыток вместо 2^{56}. Подробности можно найти в разделе 15.2.

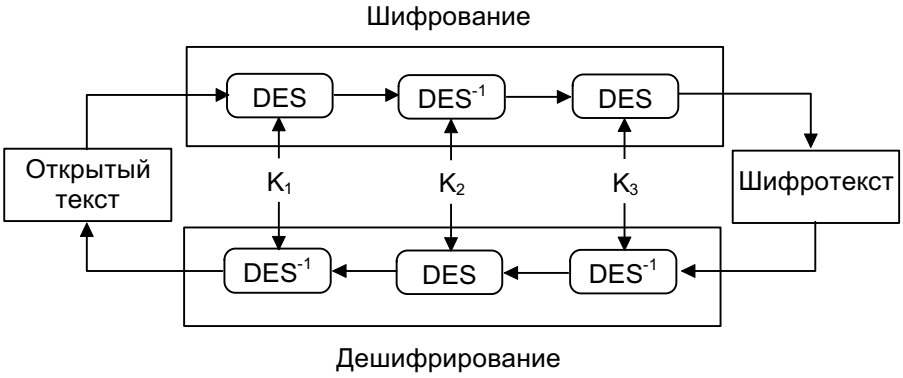


Рис. 12-10. Трехкратный DES.

DES с независимыми подключами

Другой возможностью является использование различных подключей на каждом этапе, не создавая их из одного 56-битового ключа [851]. Так как на каждом из 16 этапов используется 48 битов ключа, то длина ключа для такого варианта составит 768 битов. Такой вариант резко увеличивает сложность вскрытия алгоритма гр убой силой, сложность такого вскрытия составит 2^{768}.

Однако возможно использование вскрытия "встреча посередине" (см. раздел 15.1). Сложность такого вскр ытия уменьшается до 2^{384}, что, тем не менее, вполне достаточно для обеспечения любой мыслимой безопасн ости.

Хотя независимые подключи мешают линейному криптоанализу, этот вариант чувствителен к дифференц иальному криптоанализу и может быть вскрыт с помощью 2^{61} выбранных открытых текстов (см. -3-й) [167, 172]. По видимому, никакая модификация распределения ключей не сможет н амного усилить DES.

DESX

DESX - это вариант DES, разработанный RSA Data Security, Inc., и включенный в 1986 году в программу обеспечения безопасности электронной почты MailSafe, а в 1987 году в набор BSAFE. DESX использует метод, называемый отбеливанием (см. раздел 15.6), для маскировки входов и выходов DES. Кроме 56-битового ключа DES в DESX используется дополнительный 64-битовый ключ отбеливания. Эти 64 бита используются для в ыполнения операции XOR с блоком открытого текста перед первым этапом DES. Дополнительные 64 бита, я вляющиеся результатом применения однонаправленной функции к полному 120-битовому ключу DESX, испол ьзуются для выполнения XOR с шифротекстом, полученным в результате последнего этапа [155]. По сравнению с DES отбеливание значительно повышает устойчивость DESX к вскрытию грубой силой, вскрытие требует $(2^{120})/n$ операций при n известных открытых текстах. Также повышается устойчивость к дифференциальному и линейному криптоанализу, для вскрытия потребуется 2^{61} выбранных и 2^{60} известных открытых текстов, соответственно [1338].

CRYPT(3)

CRYPT(3) представляет собой вариант DES, используемый в системах UNIX. Он в основном используется в качестве однонаправленной функции для паролей, но иногда может быть использован и для шифрования. Ра зличие между CRYPT(3) и DES состоит в том, что в CRYPT(3) включена независимая от ключа перестановка с расширением с 2^{12} вариантами. Это сделано для того, чтобы для создания аппаратного устройства вскрытия паролей нельзя было использовать промышленные микросхемы DES.

Обобщенный DES

Обобщенный DES (Generalized DES, GDES) был спроектирован для ускорения DES и повышения устойч ивости алгоритма [1381, 1382]. Общий размер блока увеличился, а количество вычислений осталось неизме нным.

На 1-й показана поблочная диаграмма GDES. GDES работает с блоками открытого текста переменной дл ины. Блоки шифрования делятся на q 32-битовых подблоков, точное число которых зависит от полного размера блока (который по идее может меняться, но фиксирован для конкретной реализации). В общем случае q равно размеру блока, деленному на 32.

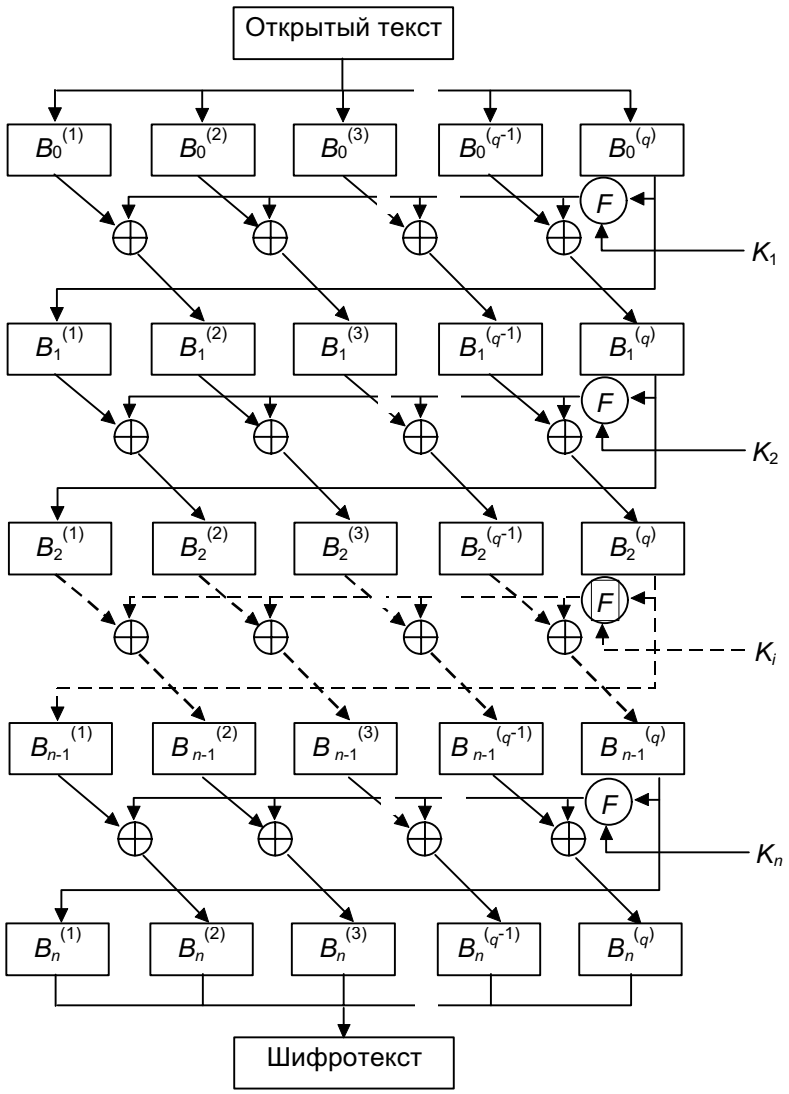


Рис. 12-11. GDES.

Функция f для каждого этапа рассчитывается один раз для крайнего правого блока. Результат при помощи операции XOR объединяется со всеми остальными частям, которые затем циклически смещаются направо. GDES использует переменное число этапов n. В последний этап внесено незначительное изменение, чтобы пр о-цессы шифрования и дешифрирования отличались только порядком подключей (точно также, как в DES). Де й-ствительно, если $q = 2$ и $n = 16$, то описанный алгоритм превращается в DES.

Бихам и Шамир [167, 168] показали, что дифференциальный криптоанализ вскрывает GDES с $q = 8$ и $n = 16$ с помощью всего шести выбранных открытых текстов. При использовании независимых подключей требуется 16 выбранных открытых текстов. GDES с $q = 8$ и $n = 22$ вскрывается с помощью всего 48 выбранных открытых текстов, а для вскрытия GDES с $q = 8$ и $n = 31$ требуется всего 500000 выбранных открытых текстов. Даже GDES с $q = 8$ и $n = 64$ слабее, чем DES - для его вскрытия нужно только 249 выбранных открытых текстов. Действительно, любая более быстрая, чем DES, схема GDES является также и менее безопасной (см. -3-й).

Недавно появился еще один вариант этой схемы [1591]. Возможно он не более безопасен, чем оригинальный GDES. Общем случае любой вариант DES с большими блоками, который быстрее DES, скорее всего менее безопасен по сравнению с DES.

DES с измененными S-блоками

Другие модификации DES связаны с S-блоками. В некоторых проектах используется переменный порядок S-блоков. Другие разработчики меняют содержание самих S -блоков. Бихам и Шамир показали [170,172], что построение S-блоков и даже их порядок оптимальны с точки зрения устойчивости к дифференциальному кри п-тоанализу:

> Изменение порядка восьми S-блоков DES (без изменения их значений) также значительно ослабляет DES: DES с 16 эт а-пами и конкретным измененным порядком вскрывается примерно за 2^{38} шагов. ... Доказано, что DES со случайными S-блоками вскрыть очень легко. Даже минимальное изменение одного из элементов S_блоков DES может снизить устойч и-

вость DES к вскрытию.

S-блоки DES не были оптимизированы против линейного криптоанализа. Существуют и лучшие S-блоки, чем предлагаемые в DES, но бездумная замена S-блоков новыми - не самая лучшая идея.

В -3-й [167, 169] перечислены некоторые модификации DES и количество выбранных открытых текстов, нужное для выполнения дифференциального криптоанализа. В таблицу не включена одна из модификаций, об ъединяющая левую и правую половины с помощью сложения по модулю 24 вместо XOR, ее в 2^{17} раз труднее вскрыть, чем DES [689].

RDES

RDES - это модификация, в которой в конце каждого этапа обмениваются местами правая и левая половины с использованием зависимой от ключа перестановки [893]. Обмены местами фиксированы и зависят только от ключа. Это означает, что может быть 15 обменов, зависимых от ключа, и 2^{15} возможных вариантов, а также что эта модификация не устойчива по отношению к дифференциальному криптоанализу [816, 894, 112]. У RDES большое количество слабых ключей. Действительно, почти каждый ключ слабее, чем типичный ключ DES. И с-пользовать эту модификацию нельзя.

Лучшей является идея выполнять обмен местами только в пределах правой половины и в начале каждого этапа. Другой хорошей идеей является выполнение обмена в зависимости от входных данных, а не как статич е-ской функции ключа. Существует множество возможных вариантов [813, 815]. В RDES-1 используется завис я-щая от данных перестановка 16-битовых слов в начале каждого этапа. В RDES-2 применяется зависящая от данных перестановка байтов в начале каждого этапа после 16-битовых перестановок, аналогичных RDES-1. Развитием этой идеи является RDES-4, и т.д. RDES-1 устойчив и к дифференциальному [815], и к линейному криптоанализу [1136]. По видимому, RDES-2 и последующие варианты достаточно хороши.

Табл. 12-15.
Вскрытия вариантов DES с помощью дифференциального криптоанализа

Изменение работы	Количество выбранных открытых текстов
Полный DES (без изменений)	2^{47}
P-перестановка	Не может усилить
Тождественная перестановка	2^{19}
Порядок S-блоков	2^{38}
Замена XOR сложениями	2^{39}, 2^{31}
S-блоки	
Случайные	2^{18} - 2^{20}
Случайные перестановки	2^{33} - 2^{41}
Одноэлементные	2^{33}
Однородные таблицы	2^{26}
Удаление E-расширения	2^{26}
Порядок E-расширения и XOR подключа	2^{44}
GDES (ширина q=8)	
16 этапов	6, 16
64 этапа	2^{49} (независимый ключ)

s"DES

Группа корейских исследователей под руководством Кванджо Кима (Kwangjo Kim) попыталась найти набор S-блоков, оптимально устойчивых и против дифференциального, и против линейного криптоанализа. Их первая попытка, известная как s^2DES, представленная в [834], оказалась, как было показано в [855, 858], менее усто й-чивой, чем DES, против дифференциального криптоанализа. Следующий вариант, s^3DES, был представлен в [839] и оказался менее устойчив, чем DES, к линейному криптоанализу [856, 1491, 1527, 858, 838]. Бихам пре д-

ложил незначительно изменить алгоритм, чтобы сделать s^3DES безопасным по отношению и к дифференциальному, и к линейному криптоанализу [165]. Исследователи вернулись к своим компьютерам и разработали улучшенную технику проектирования S-блоков [835, 837]. Они предложили s^4DES [836], а затем s^5DES [838, 944].

В -4-й приведены для s3DES (с обращенными S-блоками 1 и 2), которые безопасны по отношению к обоим видам криптоанализа. Использование этого варианта вместе с трехкратным DES наверняка помешает крипто анализу.

DES с S-блоками, зависящими от ключа

Линейный и дифференциальный криптоанализ работают только, если аналитику известно строение S-блоков. Если S-блоки зависят от ключа и выбираются криптографически сильным методом, то линейный и дифференциальный криптоанализ значительно усложнятся. Хотя надо помнить, что даже у хранящихся в секрете случайно созданных S-блоков очень плохие дифференциальные и линейные характеристики.

Табл. 12-16.
S-блоки s3DES (с обращенными S-блоками 1 и 2)

S-блок 1:															
13	14	0	3	10	4	7	9	11	8	12	6	1	15	2	5
8	2	11	13	4	1	14	7	5	15	0	3	10	6	9	12
14	9	3	10	0	7	13	4	8	5	6	15	11	12	1	2
1	4	14	7	11	13	8	2	6	3	5	10	12	0	15	9
S-блок 2:															
15	8	3	14	4	2	9	5	0	11	10	1	13	7	6	12
6	15	9	5	3	12	10	0	13	8	4	11	14	2	1	7
9	14	5	8	2,	4	15	3	10	7	6	13	1	11	12	0
10	5	3	15	12	9	0	6	1	2	8	4	11	14	7	13
S-блок 3:															
13	3	11	5	14	8	0	6	4	15	1	12	7	2	10	9
4	13	1	8	7	2	14	11	15	10	12	3	9	5	0	6
6	5	8	11	13	14	3	0	9	2	4	1	10	7	15	12
1	11	7	2	8	13	4	14	6	12	10	15	3	0	9	5
S-блок 4:															
9	0	7	11	12,	5	10	6	15	3	1	14	2	8	4	13
5	10	12	6	0	15	3	9	8	13	11	1	7	2	14	4
10	7	9	12	5	0	6	11	3	14	4	2	8	13	15	1
3	9	15	0	6	10	5	12	14	2	1	7	13	4	8	11
S-блок 5:															
5	15	9	10	0	3	14	4	2	12	7	1	13	6	8	11
6	9	3	15	5	12	0	10	8	7	13	4	2	11	14	1
15	0	10	9	3	5	4	14	8	11	1	7	6	12	13	2
12	5	0	6	15	10	9	3	7	2	14	11	8	1	4	13
S-блок 6:															
4	3	7	10	9	0	14	13	15	5	12	6	2	11	1	8
14	13	11	4	2	7	1	8	9	10	5	3	15	0	12	6
13	0	10	9	4	3	7	14	1	15	6	12	8	5	11	2
1	7	4	14	11	8	13	2	10	12	3	5	6	15	0	9

S-блок 7:

4	10	15	12	2	9	1	6	11	5	0	3	7	14	13	8
10	15	6	0	5	3	12	9	1	8	11	13	14	4	7	2
2	12	9	6	15	10	4	1	5	11	3	0	8	7	14	13
12	6	3	9	0	5	10	15	2	13	4	14	7	11	1	8

S-блок 8:

13	10	0	7	3	9	14	4	2	15	12	1	5	6	11	8
2	7	13	1	4	14	11	8	15	12	6	10	9	5	0	3
4	13	14	0	9	3	7	10	1	8	2	11	15	5	12	6
8	11	7	14	2	4	13	1	6	5	9	0	12	15	3	10

Вот как можно использовать 48 дополнительных битов ключа для создания S-блоков, устойчивых как к л инейному, так и к дифференциальному криптоанализу [165].

(1) Изменить порядок S-блоков DES: 24673158.

(2) Выбрать 16 из оставшихся битов ключа. Если первый бит 1, обменять местами первые и последние два ряда S-блока 1. Если второй бит 1, обменять местами первые и последние восемь столбцов S-блока 1. П овторить то же самое для третьего и четвертого битов и S-блока 2. Повторить то же самое для S-блоков с 3 по 8.

(3) Взять оставшиеся 32 бита ключа. Выполнить XOR первых четырех битов с каждым элементом S-блока 1, XOR следующих четырех битов с каждым элементом S-блока 2, и так далее.

Сложность вскрытия такой системы с помощью дифференциального криптоанализа составит 251, с пом ощью линейного криптоанализа - 2^{53}. Сложность исчерпывающего перебора составит 2102.

Что хорошо в этом варианте DES так это то, что он может быть реализован в существующей аппаратуре. Различные поставщики микросхем DES продают микросхемы DES с возможностью загрузки S-блоков. Можно реализовать любой способ генерации S-блоков вне микросхемы и затем загрузить их в нее. Для дифференц иального и линейного криптоанализа нужно так много известных или выбранных открытых текстов, что эти сп особы вскрытия становятся неосуществимыми. Вскрытие грубой силой также трудно себе представить, не пом ожет никакое увеличение скорости.

12.7 Насколько безопасен сегодня DES?

Ответ одновременно и прост, и труден. При простом ответе учитывается только длина ключа (см. раздел 7.1). Машина для вскрытия DES грубой силой, способная найти ключ в среднем за 3.5 часа, в 1993 году стоила 1 миллион долларов [1597, 1598]. DES используется очень широко, и наивно было бы предполагать, что NSA и аналогичные организации в других странах не построили по такому устройству. И не забывайте, что стоимость уменьшается в 5 раз каждые 10 лет. С течением времени DES будет становиться все менее и менее безопасным.

Для трудного ответа нужно попытаться оценить криптоаналитические методы. Дифференциальный крипто анализ был известен в NSA задолго до середины 70-х, когда DES впервые стал стандартом. Наивно считать, что с тех пор теоретики NSA ничего не делали, почти наверняка они разработали новые криптоаналитические мет оды, которые можно использовать против DES. Но фактов у нас нет, одни слухи.

Винн Шварцтау (Winn Schwartau) пишет, что NSA построило огромную параллельную машину для вскр ытия DES уже в середине 80-х [1404]. По крайней мере одна такая машина была построена в Harris Corp. С и спользованием Cray Y-MP. Предположительно существует ряд алгоритмов, которые на несколько порядков уменьшают сложность вскрытия DES грубой силой. Контекстные алгоритмы, основанные на внутренней работе DES, позволяют отбросить ряд ключей, используя частичные решения. Статистические алгоритмы уменьшают эффективную длину ключа еще сильнее. Другие алгоритмы также проверяют вероятные ключи - слова, печ атаемые последовательности ASCII, и т.д. (см. раздел 8.1). По слухам NSA может вскрыть DES за время от 3 до 15 минут, в зависимости от того коков будет выполненный объем предварительной обработки. И каждая такая машина стоит порядка 50000 долларов.

Согласно другим слухам, если у NSA есть большое количество открытых текстов и шифротекстов, его эк сперты могут выполнить некоторые статистические расчеты и затем считать ключ из архива на оптических ди сках.

И то, что это только слухи, не дает мне чувство уверенности в DES. Этот алгоритм очень долго был очень большой мишенью. Почти любое изменение DES послужит дополнительной защитой, может быть получивши йся шифр и будет менее устойчив к вскрытию, но у NSA может не оказаться средств решения этой конкретной задачи.

Я рекомендую использовать схему Бихама для зависящих от ключа S-блоков. Она может быть легко реал изована программно или аппаратно (с помощью микросхем с загружаемыми S-блоками), и не приводит к потере эффективности по сравнению с DES. Эта схема повышает устойчивость алгоритма к вскрытию грубой силой, усложняет дифференциальный и линейный криптоанализ и заставляет NSA столкнуться с алгоритмом, по кра йней мере таким же сильным как DES, но другим.

Глава 13 Другие блочные шифры

13.1 LUCIFER

В конце 60-х IBM начала выполнение исследовательской программы по компьютерной криптографии, наз ыванной Люцифером (Lucifer) и руководимой сначала Хорстом Фейстелем (Horst Feistel), а затем Уолтом Тачм еном (Walt Tuchman). Это же название - Lucifer - получил блочный алгоритм, появившийся в результате этой программыв начале 70-х [1482, 1484]. В действительности существует по меньшей мере два различных алг оритма с таким именем [552, 1492]. [552] содержит ряд пробелов в спецификации алгоритма. Все это привело к заметной путанице.

Lucifer - это набор перестановок и подстановок, его блоки похожи на блоки DES. В DES результат функции f объединяется с помощью XOR со входом предыдущего этапа, образуя вход следующего этапа. У S-блоков алг оритма Lucifer 4-битовые входы и 4-битовые выходы, вход S-блоков представляет собой перетасованный выход S_блоков предыдущего этапа, входом S-блоков первого этапа является открытый текст. Для выбора использу емого S-блока из двух возможных применяется бит ключа. (Lucifer реализует это, как один T-блок с 9 битами на входе и 8 битами на выходе.) В отличие от DES половины блока между этапами не переставляются и вообще понятие половины блока не используется в алгоритме Lucifer. У этого алгоритма 16 этапов, 128-битовые блоки и более простое, чем в DES, распределение ключей.

Применив дифференциальный криптоанализ к первой реализации Lucifer'a, Бихам и Шамир [170, 172] пок азали, что Lucifer с 32-битовыми блоками и 8 этапами может быть взломан с помощью 40 выбранных открытых текстов за 2^{39} шагов, тот же способ позволит вскрыть Lucifer с 128-битовыми блоками и 8 этапами с помощью 60 выбранных открытых текстов за 2^{53} шагов. 18-этапный, 128-битовый Lucifer вскрывается дифференциал ьным криптоанализом с помощью 24 выбранных открытых текстов за 2^{21} шагов. Все эти вскрытия использовали сильные S-блоки DES. Применив дифференциальный криптоанализ против второй реализации Lucifer, Бихам и Шамир обнаружили, что S-блоки намного слабее, чем в DES. Дальнейший анализ показал, что более половины возможных ключей не являются безопасными [112]. Криптоанализ со связанными ключами может взломать 128-битовый Lucifer с любым числом этапов с помощью 2^{33} выбранных открытых текстов для выбранных кл ючей или 2^{65} известных открытых текстов для выбранных ключей [158]. Вторая реализация Lucifer еще слабее [170, 172, 112].

Некоторые думают, что Lucifer безопаснее, чем DES, из-за большей длины ключа и малого количества опу бликованных сведений. Но очевидно, что это не так.

Lucifer является объектом нескольких патентов США: [553, 554, 555, 1483]. Сроки действия всех этих п атентов истекли.

13.2 MADRYGA

В.Е. Мадрига (W. E. Madryga) предложил этот блочный алгоритм в 1984 году [999]. Он может быть эффе ктивно реализован как программа: в нем нет надоедливых перестановок, и все операции выполняются над ба йтами. Стоит перечислить задачи, которые решал автор при проектировании алгоритма:

1. Открытый текст нельзя получить из шифротекста без помощи ключа. (Это означает только то, что а лгоритм безопасен.)
2. Количество операций, нужное для определения ключа по имеющимся шифротексту и открытому те ксту, должно быть статистически равно произведению количества операций при шифровании на число возможных ключей. (Это означает, что никакое вскрытие с открытым текстом не может быть лучше, чем вскрытие грубой силой.)
3. Известность алгоритма не влияет на силу шифра. (Безопасность полностью опр еделяется ключом.)
4. Изменение одного бита ключа должно вызывать для того же открытого текста радикальное изменение шифротекста, и Изменение одного бита открытого текста должно вызывать для того же ключа рад икальное изменение шифротекста. (Это лавинный э ффект.)
5. Алгоритм должен содержать некоммутативную комбинацию подстановок и перестан овок.
6. Подстановки и перестановки, используемые в алгоритме, должны определяться и входными данными, и ключом.
7. Избыточные группы битов открытого текста должны быть полностью замаскированы в шифротексте.
8. Длина шифротекста должна равняться длине открытого текста.

9. Не должно быть простых взаимосвязей между любыми возможными ключами и особенностями ши фротекста.
10. Все возможные ключи должны давать сильный шифр. (Не должно быть слабых кл ючей.)
11. Длина ключа и текста могут регулироваться для реализации различных требований к безопасности.
12. Алгоритм должен позволять эффективную программную реализацию на больших мэйнфреймах, м иникомпьютерах, микрокомпьютерах и с помощью дискретной логики. (По сути используемые в алг оритме функции ограничены XOR и битовым сдвигом.)

DES удовлетворял первым девяти требованиям, но последние три были новыми. В предположении, что лу чшим способом вскрытия алгоритма является грубая сила, переменная длина ключа, конечно же, заставит з амолчать тех, кто считает, что 56 битов - это слишком мало. Такие люди могут реализовать этот алгоритм с л юбой нужной им длиной ключа. А любой, кто когда-нибудь пытался реализовать DES программно, обрадуется алгоритму, который учитывает возможности программных реал изаций.

Описание Madryga

Madryga состоит из двух вложенных циклов. Внешний цикл повторяется восемь раз (но это количество м ожет быть увеличено для повышения) и содержит применение внутреннего цикла к открытому тексту. Внутре нний цикл превращает открытый текст в шифротекст, повторяясь для каждого 8-битового блока (байта) открыт ого текста. Следовательно, весь открытый текст восемь раз последовательно обрабатывается а лгоритмом.

Итерация внутреннего цикла оперирует с 3-байтовым окном данных, называемым рабочим кадром (см. 12-й). Это окно смещается на 1 байт за итерацию. (При работе с последними 2 байтами данные считаются цикл ически замкнутыми.) Первые два байта рабочего кадра циклически сдвигаются на переменное число позиций, а для последнего байта выполняется XOR с некоторыми битами ключа. По мере продвижения рабочего кадра все байты последовательно "вращаются" и подвергаются операции XOR с частями ключа. Последовательные вр ащения перемешивают результаты предыдущих операций XOR и вращения, а результат XOR влияет на вращ ение. Это делает весь процесс обратимым.

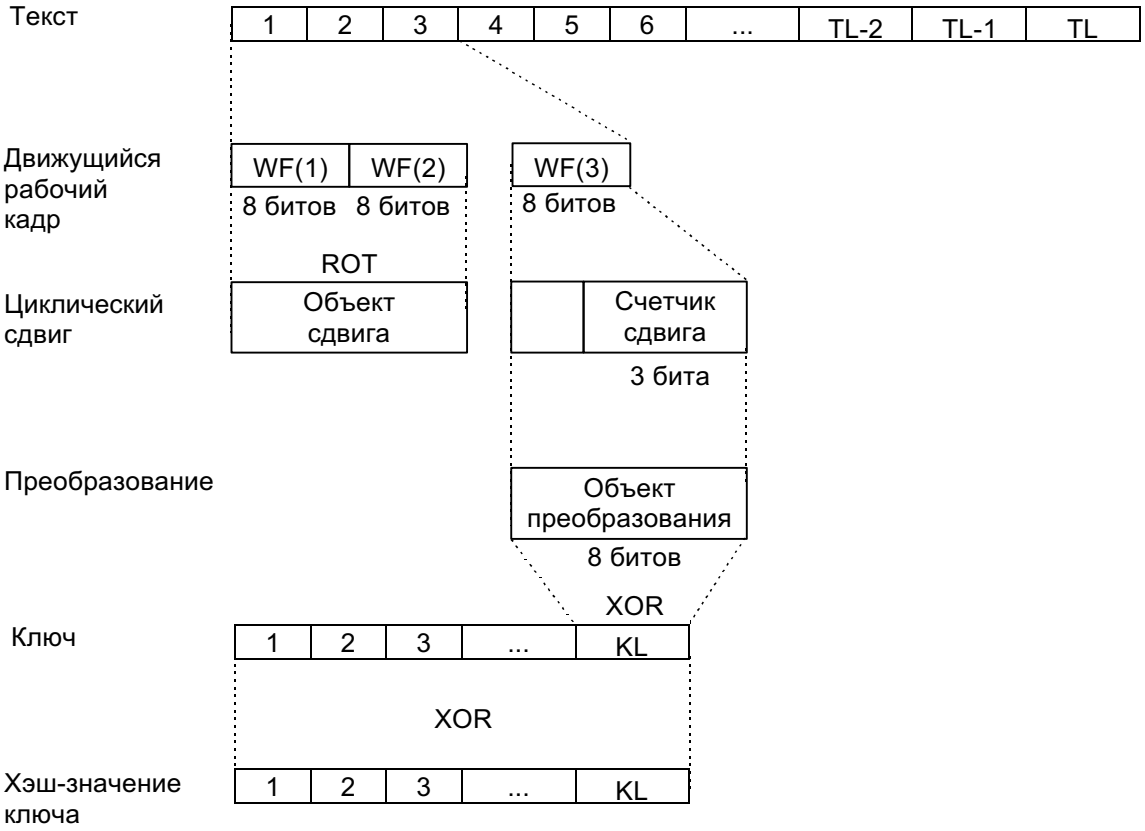


Рис. 13-1. Одна итерация Madryga.

Так как каждый байт данных влияет на два байта слева от себя и на один байт справа, после восьми прох одов каждый байт шифротекста зависит от 16 байтов слева и от восьми байтов справа.

При шифровании каждая итерация внутреннего цикла устанавливает рабочий кадр на предпоследний байт открытого текста и циклически перемещает его к байту открытого текста, третьему слева от последнего. Снач а-

ла весь ключ подвергается операции XOR со случайной константой и затем циклически смещается влево на 3 бита. Младшие три бита младшего байта рабочего кадра сохраняются, они определяют вращение остальных двух байтов. Затем для младшего байта рабочего кадра выполняется операция XOR с младшим байтом ключа. Далее объединение двух старших байтов циклически смещается влево на переменное число битов (от 0 до 7). Наконец рабочий кадр смещается вправо на один байт и весь процесс повторяется.

Смысл случайной константы в том, чтобы превратить ключ в псевдослучайную последовательность. Длина константы должна быть равна длине ключа. При обмене данными абоненты должны пользоваться константой одинаковой длины. Для 64-битового ключа Мадрига рекомендует константу 0x0f1e2d3c4b5a6978.

При дешифрировании процесс инвертируется. При каждой итерации внутреннего цикла рабочий кадр уст анавливается на байт, третий слева от последнего байта шифротекста, и циклически перемещается в обратном направлении до байта, который находится на 2 байта левее последнего байта шифротекста. И ключ, и 2 байта шифротекста в процессе циклически смещаются направо, а XOR выполняется перед циклическими сдвигами.

Криптоанализ и Madryga

Исследователи из Технического университета в Квинсланде (Queensland University of Technology) [675] и с следовали Madryga вместе с некоторыми другими блочными шифрами. Они обнаружили, что в этом алгоритме не проявляется лавинный эффект для преобразования открытого текста в шифротекст. Кроме того, во многих шифротекстах процент единиц был выше, чем процент нулей.

Хотя у меня нет сведений о проведении формального анализа этого алгоритма, он не производит впечатл ение супернадежного. При поверхностном знакомстве с ним Эли Бихам пришел к следующим выводам [160]:

> Алгоритм состоит только из линейных операций (циклическое смещение и XOR), незначительно изменяемых в завис имости от данных.
>
> В этом нет ничего похожего на мощь S-блоков DES.
>
> Четность всех битов шифротекста и открытого текста неизменна и зависит только от ключа. Поэтому, обладая открытым текстом и соответствующим шифротекстом, можно предсказать четность шифротекста для любого открытого текста.

По отдельности ни одно из этих замечаний не являются критическими, но этот алгоритм не вызывает у меня положительных эмоций. Я не рекомендую использовать Madryga.

13.3 NewDES

NewDES (новый DES) был спроектирован в 1985 году Робертом Скоттом (Robert Scott) как возможная зам ена DES [1405, 364]. Алгоритм не является модификацией DES, как может показаться из его названия. Он оп ерирует 64-битовыми блоками шифротекста, но использует 120-битовый ключ. NewDES проще, чем DES, в нем нет начальной и заключительной перестановок. Все операции выполняются над целыми байтами. (На самом деле NewDES ни коим образом не является новой версией DES, название было выбрано неудачно.)

Блок открытого текста делится на восемь 1-байтовых подблоков: B_0, B_1, . . ., B_6, B_7. Затем подблоки проходят через 17 этапов. В каждом этапе восемь действий. В каждом действии один из подблоков подвергается опер ации XOR с частью ключа (есть одно исключение), заменяется другим байтом с помощью функции f и затем подвергается операции XOR с другим подблоком, который и заменяется результатом. 120-битовый ключ дели тся на 15 подблоков ключа: K_0, K_1, . . ., K_{13}, K_{14}. Процесс легче понять, увидев его схему, чем прочитав его оп исание. Алгоритм шифрования NewDES показан на 11-й.

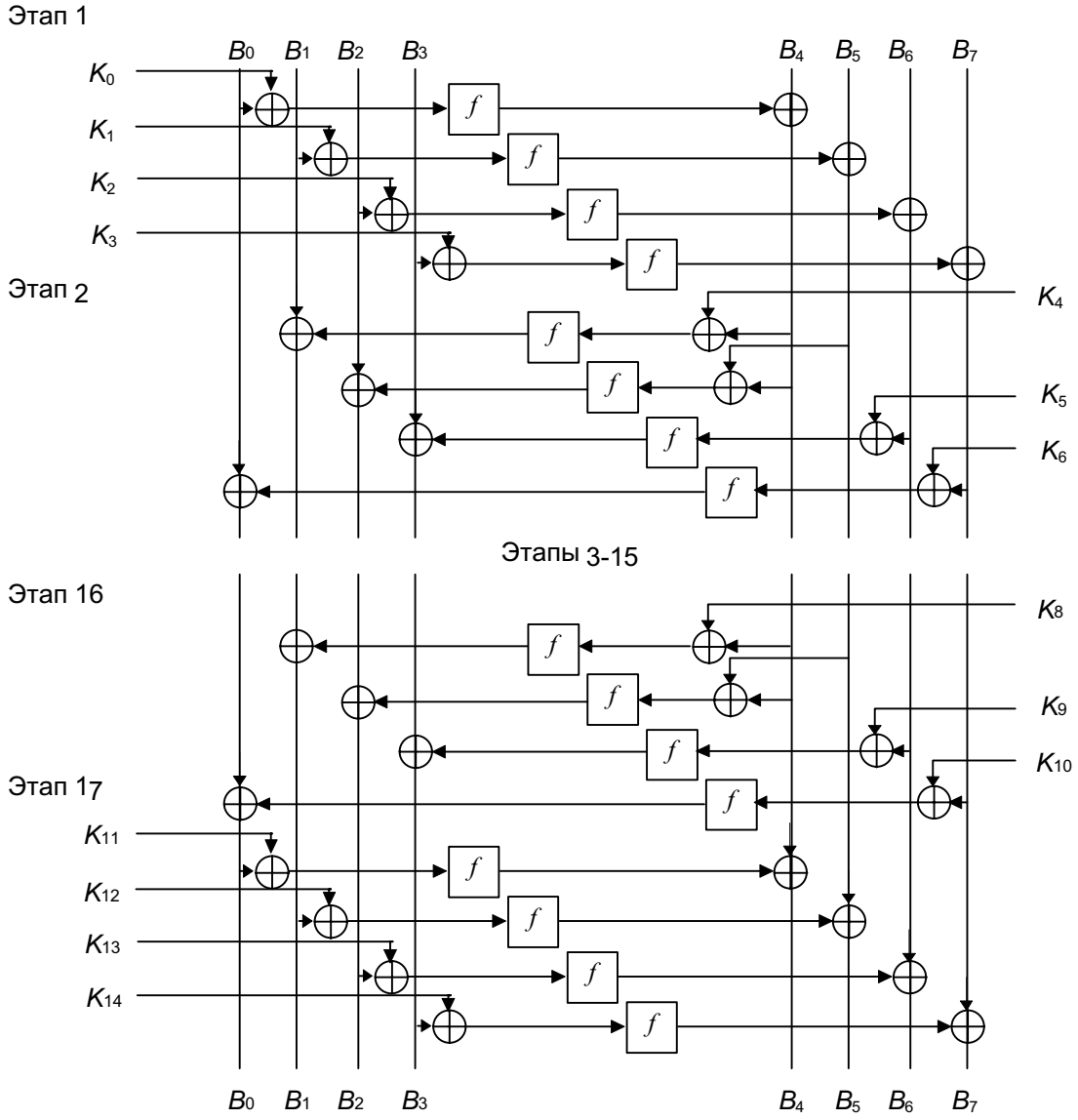


Рис. 13-2. NewDES.

Функция f выводится из Декларации независимости. Подробности можно найти в [1405].

Скотт показал, что каждый бит блока открытого текста влияет на каждый бит шифротекста уже после 7 этапов. Он также проанализировал функцию f и не нашел каких-либо очевидных проблем. NewDES обладает той же комплиментарностью, что и DES [364]: если $E_K(P\} = C$, то $E_K(P'\} = C'$. Это уменьшает объем работы, необходимой для вскрытия грубой силой, с 2^{110} действий до 2^{119}. Бихам заметил, что любое изменение полного байта, примененное ко всем байтам ключа и данных, также приводит к комплиментарности [160]. Это уменьшает объем грубого вскрытия до 2^{112} действий.

Это не является критичным, но предложенное Бихамом криптоаналитическое вскрытие со связанными ключами может вскрыть NewDES с помощью 2^{33} выбранных открытых текстов для выбранных ключей за 2^{48} действий [160]. Хотя такое вскрытие требует много времени и в большой степени является теоретическим, оно показывает, что NewDES слабее, чем DES.

13.4 FEAL

FEAL был предложен Акихиро Шимузу (Akihiro Shimizu) Шоджи Миягучи (Shoji Miyaguchi) из NTT Japan [1435]. В нем используются 64-битовый блок и 64-битовый ключ. Его идея состоит в том, чтобы создать алгоритм, подобный DES, но с более сильной функцией этапа. Используя меньше этапов, этот алгоритм мог бы работать быстрее. К несчастью действительность оказалась далека от целей проекта.

Описание FEAL

На 10-й представлена блок-схема одного этапа FEAL. В качестве входа процесса шифрования используется 64-битовый блок открытого текста. Сначала блок данных подвергается операции XOR с 64 битами ключа. З а-

тем блок данных расщепляется не левую и правую половины. Объединение левой и правой половин с помощью XOR образует новую правую половину. Левая половина и новая правая половина проходят через n этапов (первоначально четыре). На каждом этапе правая половина объединяется с помощью функции f с шестнадцатью битами ключа и с помощью XOR - с левой половиной, создавая новую правую половину. Исходная правая п о-ловина (на начало этапа) становится новой левой половиной. После n этапов (не забывайте, что левая и правая половины не переставляются после n-го этапа) левая половина снова объединяется с помощью XOR с правой половиной, образуя новую правую половину, затем левая и правая соединяются вместе в 64-битовое целое. Блок данных объединяется с помощью XOR с другими 64 битами ключа, и алгоритм завершается.

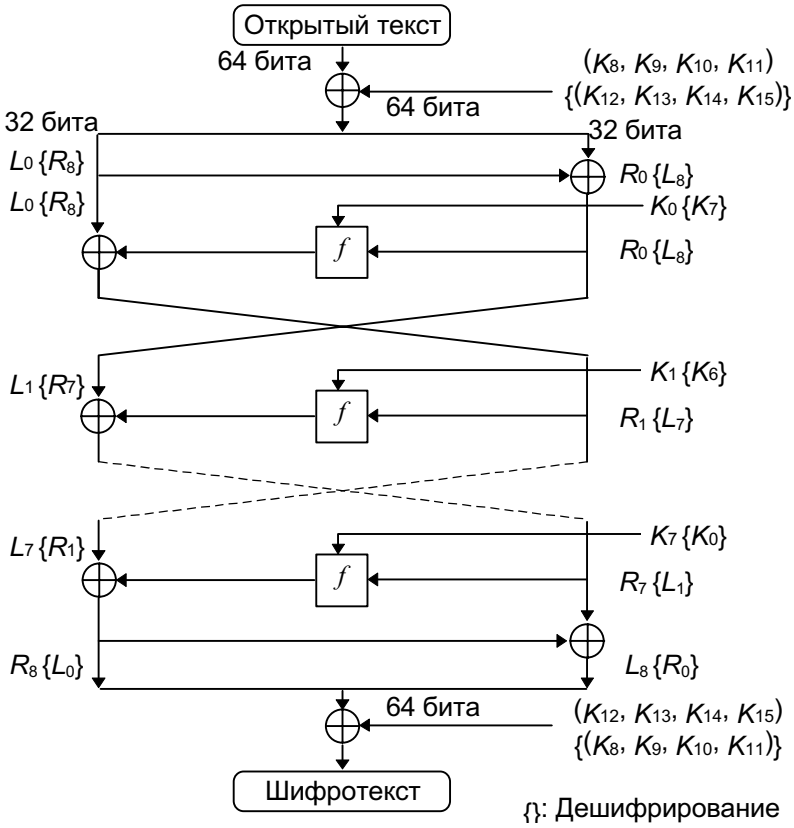


Рис. 13-3. Один этап FEAL.

Функция f берет 32 бита данных и 16 битов ключа и смешивает их вместе. Сначала блок данных разбивается на 8-битовые кусочки, которые затем объединяются с помощью XOR и заменяют друг друга. Блок-схема фун к-ции f представлена на 9-й. Две функции S_0и S_1 определяются следующим образом:

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

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

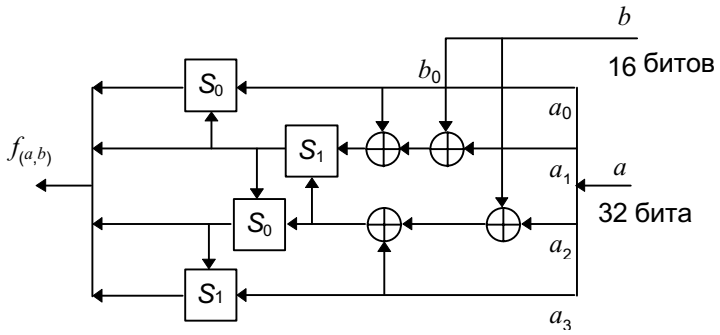


Рис. 13-4. Функция f.

Тот же алгоритм может быть использован для дешифрирования. Единственным отличием является то, что при дешифрировании порядок использования частей ключа меняется на обратный.

На 8-й представлена блок-схема функции генерации ключа. Сначала 64-битовый ключ делится на две пол о-

вины, к которым применяются операции XOR и функции f_k, как показано на схеме. На 7-й показана блок-схема функции f_k. Два 32-битовых входа разбиваются на 8-битовые блоки, объединяемые и заменяемые в соответс твии со схемой. S_0 и S_1 определяются, как показано на рисунке. Затем в алгоритме шифрования/дешифрирования используются 16-битовые блоки ключа.

На микропроцессоре 80286/10 МГц ассемблерная реализация FEAL-32 может шифровать данные со скор остью 220 Кбит/с. FEAL-64 может шифровать данные со скоростью 120 Кбит/с [1104].

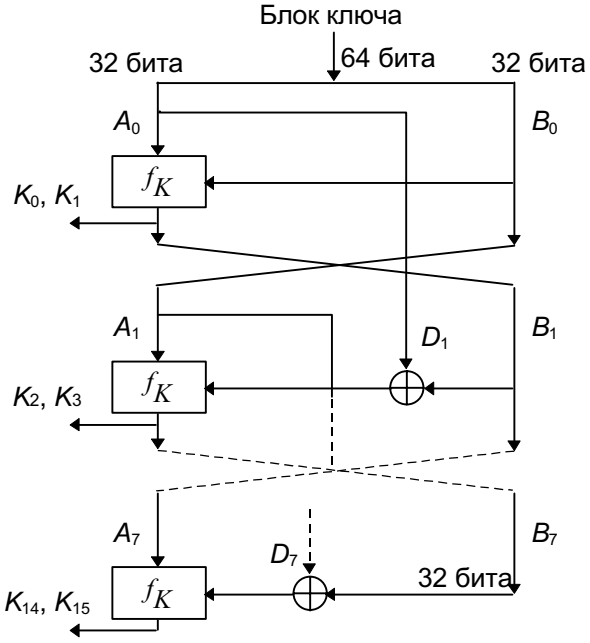
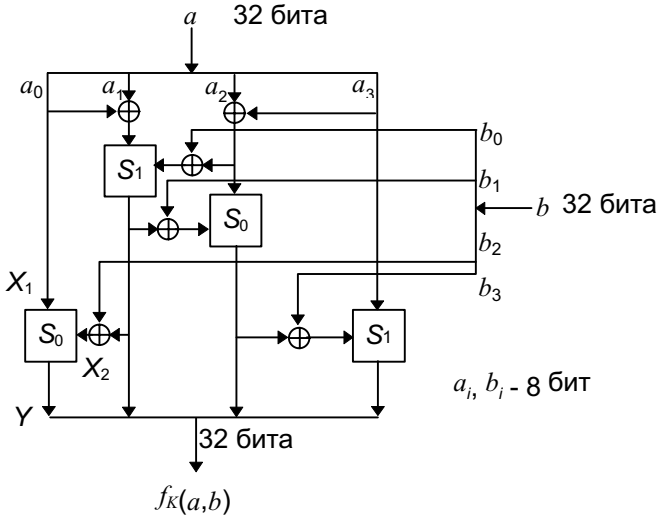


Рис. 13-5. Обработка ключа в FEAL.



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

Рис. 13-6. Функция f_K.

Криптоанализ FEAL

Успешный криптоанализ FEAL-4, FEAL с четырьмя этапами, был выполнен с помощью вскрытия с выбра нными открытыми текстами [201], а позже слабость этого алгоритма была показана в [1132]. Последнее вскр ытие, выполненное Сином Мерфи (Sean Murphy), было первым опубликованным вскрытием, использовавшим дифференциальный криптоанализ, и для него потребовалось только 20 выбранных открытых текстов. Ответом разработчиков стал 8-этапный FEAL [1436, 1437, 1108], криптоанализ которого был представлен Бихамом и

Шамиром на конференции SECURICOM '89 [1424]. Для вскрытия FEAL-8 с выбранными открытыми текстами потребовалось только 10000 блоков [610], что заставило разработчиков алгоритма засучить рукава и опред елить FEAL-N [1102, 1104], алгоритм с переменным числом этапов (конечно же, большим 8).

Бихам и Шамир применили против FEAL-N дифференциальный криптоанализ, хотя они могли бы еще б ыстрее вскрыть его грубой силой (с помощью менее, чем 2^{64} шифрований выбранного открытого текста) для N , меньшего 32. [169]. Для вскрытия FEAL-16 нужно 2^{28} выбранных или $2^{46.5}$ известных открытых текстов. Для вскрытия FEAL-8 требуется 2000 выбранных или $2^{37.5}$ известных открытых текстов. FEAL-4 может быть вскрыт с помощью всего 8 правильно выбранных открытых текстов.

Разработчики FEAL определили также модификацию FEAL - FEAL-NX, в которой используется 128-битовый ключ (см. 6-й) [1103, 1104]. Бихам и Шамир показали, что для любого значения N FEAL-NX со 128-битовым ключом взламывать не сложнее, чем FEAL-N с 64-битовым ключом [169]. Недавно был предложен FEAL-N(X)S, усиливающий FEAL за счет динамической фун кции обмена местами [1525].

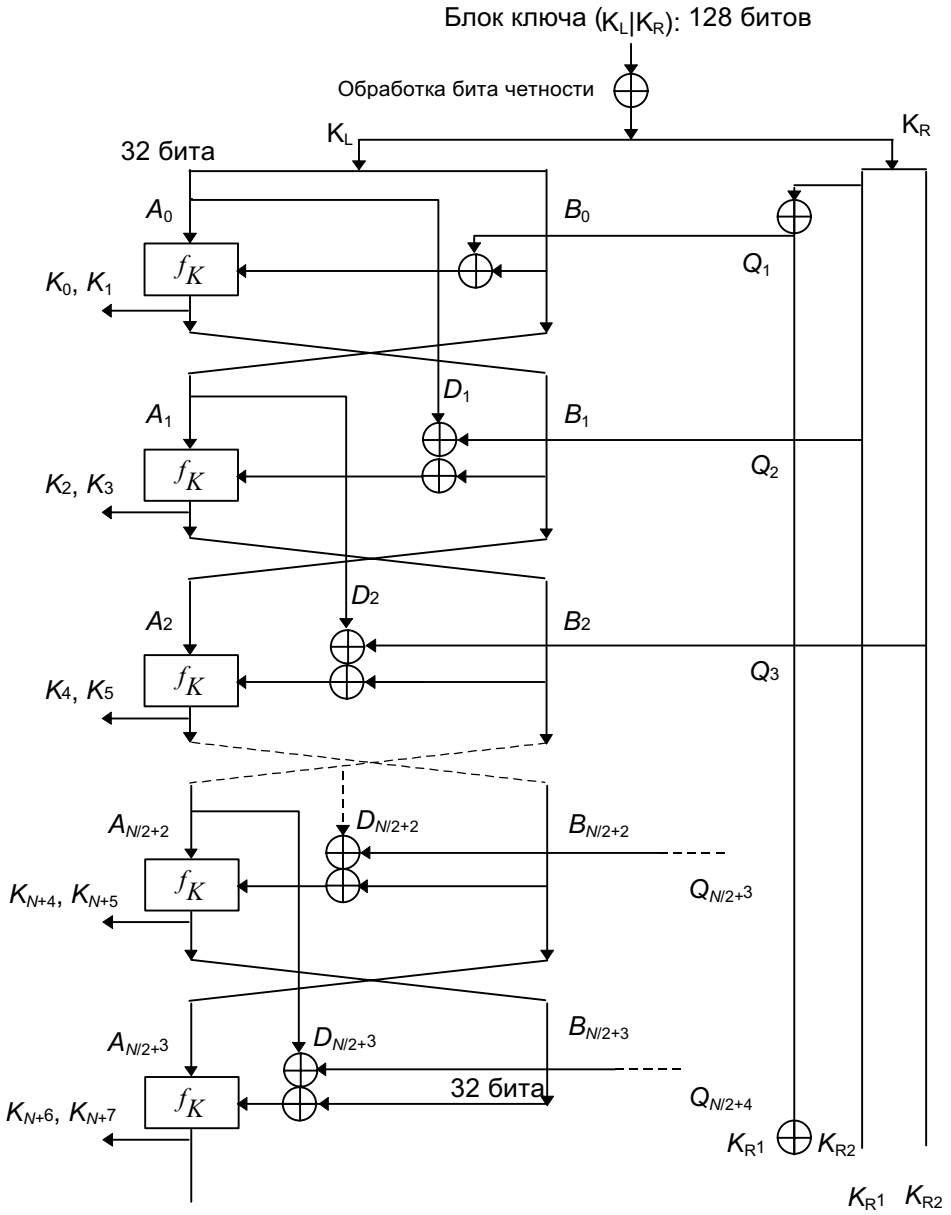


Рис. 13-7. Обработка ключа в FEAL-NX.

Более того. В [1520] было представлено другое вскрытие FEAL-4, требующее только 1000 известных откр ытых текстов, и FEAL-8, для которого нужно только 20000 известных открытых текстов. Другие вскрытия прив едены в [1549, 1550]. Наилучшим является выполненное Мицуру Мацуи (Mitsuru Matsui) и Атшуиро Ямагиши

(Atshuiro Yamagishi) [1020]. Это было первое применение линейного криптоанализа, и оно позволило вскрыть FEAL-4 с помощью 5 известных открытых текстов, FEAL-6 - с помощью 100 известных открытых текстов, а FEAL-8 - с помощью 2^{15} известных открытых текстов. Дальнейшие уточнения можно найти в [64]. Дифференциальный криптоанализ позволяет вскрывать FEAL-8, используя только 12 выбранных открытых текстов [62]. Кто бы не изобрел новый метод криптоаналитического вскрытия, кажется, что он всегда сначала пробует его на FEAL.

Патенты

FEAL запатентован в Соединенных Штатах [1438], соответствующие патенты приняты к рассмотрению в Англии, Франции и Германии. Желающий лицензировать использование алгоритма должен связаться с Департаментом интеллектуальной собственности (Intellectual Property Department), NTT, 1-6 Uchisaiwai-cho, 1-chome, Chiyada-ku, 100 Japan.

13.5 REDOC

REDOC II представляет собой другой блочный алгоритм, разработанный Майклом Вудом (Michael Wood) для Cryptech, Inc. [1613, 400]. В нем используются 20-байтовый (160-битовый) ключ и 80-битовый блок.

REDOC II выполняет все манипуляции - перестановки, подстановки и XOR с ключом - с байтами, этот алгоритм эффективен при программной реализации. REDOC II использует меняющиеся табличные функции. В отличие от DES, имеющего фиксированный (хотя и оптимизированных для безопасности) набор таблиц подстановок и перестановок REDOC II использует зависимые от ключа и открытого текста наборы таблиц (по сути S-блоков). У REDOC II 10 этапов, каждый этап представляет собой сложную последовательность манипуляций с блоком.

Другой уникальной особенностью является использование **масок**, которые являются числами, полученными из таблицы ключей, и используются для выбора таблиц данной функции для данного этапа. Для выбора таблиц функции используются как значение данных, так и маски.

При условии, что самым эффективным средством вскрытия этого алгоритма является грубая сила, REDOC II очень надежен: для вскрытия ключа требуется 2^{160} операций. Томас Кузик (Thomas Cusick) выполнил криптоанализ одного этапа REDOC II, но ему не удалось расширить вскрытие на несколько этапов [400]. Используя дифференциальный криптоанализ, Бихам и Шамир достигли успеха в криптоанализе одного этапа REDOC II с помощью 2300 выбранных открытых текстов [170]. Они не смогли расширить это вскрытие на несколько этапов, но им удалось получить три значения маски после 4 этапов. О других попытках криптоанализа мне не известно.

REDOC III

REDOC представляет собой упрощенную версию REDOC II, также разработанную Майклом Вудом [1615]. Он работает с 80-битовым блоком. Длина ключа может меняться и достигать 2560 байтов (20480 битов). Алгоритм состоит только из операций XOR для байтов ключа и открытого текста, перестановки или подстановки не используются.

(1) Создать таблицу ключей из 256 10-байтовых ключей, используя секретный ключ.

(2) Создать 2 10-байтовых блока маски M_1 и M_2. M_1 представляет собой XOR первых 128 10-байтовых ключей, а M_2 - XOR вторых 128 10-байтовых ключей.

(3) Для шифрования 10-байтового блока:

   (a) Выполнить XOR для первого байта блока данных и первого байта M_1. Выбрать ключ из таблицы ключей, рассчитанной на этапе (1). Использовать вычисленное значение XOR в качестве индекса таблицы. Выполнить XOR каждого, кроме первого, байта блока данных с соответствующим байтом выбранного ключа.

   (b) Выполнить XOR для второго байта блока данных и второго байта M_1. Выбрать ключ из таблицы ключей, рассчитанной на этапе (1). Использовать вычисленное значение XOR в качестве индекса таблицы. Выполнить XOR каждого, кроме второго, байта блока данных с соответствующим байтом выбранного ключа.

   (c) Продолжать для всего блока данных (для байтов с 3 по 10), пока каждый байт не будет использован для выбора ключа из таблицы после выполнения для него XOR с соответствующим значением M_1. Затем выполнить XOR с ключом для каждого, кроме использованного для выбора ключа, байта.

   (d) Повторить для M_2 этапы (a)-(c).

Этот алгоритм несложен и быстр. На 33 мегагерцовом процессоре 80386 он шифрует данные со скоростью

2.75 Мбит/с. Вуд оценил, что конвейеризированная реализация на СБИС с 64 битовой шиной данных могла бы шифровать данные со скоростью свыше 1.28 Гбит/с при тактовой частоте 20 МГц.

REDOC III не безопасен [1440]. Он чувствителен к дифференциальному криптоанализу. Для восстановления обеих масок нужно всего примерно 223 выбранных открытых текстов.

Патенты и лицензии

Обе версии REDOC запатентованы в Соединенных штатах [1614]. Рассматриваются и иностранные патенты. При заинтересованности в REDOC II или REDOC III обращайтесь к Майклу Вуду (Michael C. Wood, Delta Computec, Inc., 6647 Old Thompson Rd., Syracuse, NY 13211).

13.6 LOKI

LOKI разработан в Австралии и впервые был представлен в 1990 году в качестве возможной альтернативы DES [273]. В нем используются 64-битовый блок и 64-битовый ключ. Общая структура алгоритма и использ о-вания ключа описана в [274, 275], а схема S-блоков - в [1247].

Используя дифференциальный криптоанализ, Бихам и Шамир смогли взломать LOKI с 11 и менее этапами быстрее, чем грубой силой [170]. Более того, алгоритм обладает 9-битовой комплиментарностью, что уменьш а-ет сложность вскрытия грубой силой в 256 раз [170, 916, 917].

Ларс Кнудсен (Lars Knudsen) показал, что LOKI с 14 и менее этапами чувствителен к дифференциальному криптоанализу [852, 853]. Кроме того, если в LOKI используются альтернативные S-блоки, получающийся шифр вероятно также будет чувствителен к дифференциальному криптоанализу.

LOKI91

В ответ на эти вскрытия разработчики LOKI вернулись за чертежную доску и пересмотрели свой алгоритм. Результатом было появление LOKI91 [272]. (Предыдущая версия LOKI была переименована в LOKI89.)

Чтобы повысить устойчивость алгоритма к дифференциальному криптоанализу и избавиться от комплиме н-тарности, в оригинальный проект были внесены следующие изменения:

1. Алгоритм генерации подключей был изменен так, чтобы половины переставлялись не после каждого, а после каждого второго этапа.
2. Алгоритм генерации подключей был изменен так, чтобы количество позиций циклического сдвига л е-вого подключа было равно то 12, то 13 битам.
3. Были устранены начальная и заключительная операции XOR блока и ключа.
4. Была изменена функция S-блока с целью сгладить XOR профили S-блоков (чтобы повысить их усто й-чивость к дифференциальному криптоанализу), и не допустить, чтобы для какого-то значения выпо л-нялось f(x) = 0, где f - это комбинация E-, S- и P-блоков.

Описание LOKI91

Механизм LOKI91 похож на DES (см. Рис. 13-8). Блок данных делится на левую и правую половины и пр о-ходит через 16 этапов, что очень походе на DES. На каждом этапе правая половина сначала подвергается оп е-рации XOR с частью ключа, а затем над ней выполняется перестановка с расширением (см. Табл. 13-1).

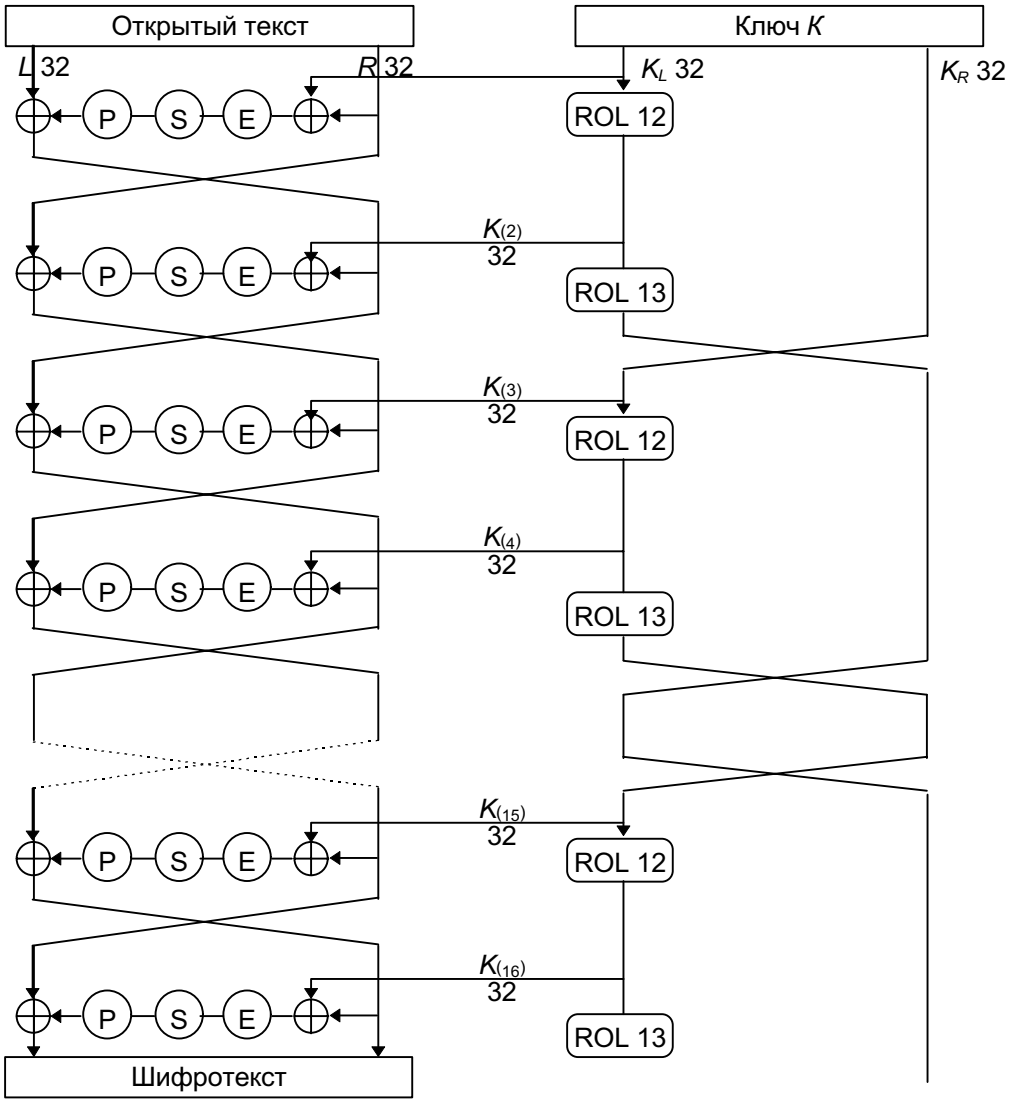


Рис. 13-8. LOKI91.

Табл. 13-1.
Перестановка с расширением

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

48-битовый результат делится на четыре 12-битовых блока, для каждого из которых выполняется следующая подстановка с использованием S-блока: берется каждый 12-битовый вход, по 2 крайних левых и крайних пр а-вых бита используются для получения номера r, в 8 центральных бит образуют номер c. Результатом S-блока - *O* - является следующее значение:

$O(r,c) = (c + ((r * 17) \oplus 0xff) \,\&\, 0xff)^{31} \bmod P_r$

P_r приведено в Табл. 13-2.

Табл. 13-2.
P_r

r:	1,	2,	3,	4,	5,	6,	7,	8,	9,	10,	11,	12,	13,	14,	15,	16
P_r:	375,	279,	391,	395,	397,	415,	419,	425,	433,	445,	451,	463,	471,	477,	487,	488

Затем четыре 8-битовых результата снова объединяются, образуя 32-битовое число, которое подвергается операции перестановки, описанной в Табл. 13-3. Наконец для получения новой левой половины выполняется XOR правой половины с прежней левой половиной, а левая половина становится новой правой половиной. П осле 16 этапов для получения окончательного шифротекста снова выполняется XOR блока и ключа.

Табл. 13-3.
Перестановка с помощью P-блока

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

Подключи из ключа выделяются достаточно прямолинейно. 64-битовый ключ разбивается на левую и пр авую половины. На каждом этапе подключом является левая половина. Далее она циклически сдвигается влево на 12 или 13 битов, затем после каждых двух этапов левая и правая половины меняются местами. Как и в DES для шифрования и дешифрирования используется один и тот же алгоритм с некоторыми изменениями в испол ьзовании подключей.

Криптоанализ LOKI91

Кнудсен предпринял попытку криптоанализа LOKI91 [854, 858], но нашел, что этот алгоритм устойчив к дифференциальному криптоанализу. Однако ему удалось обнаружить, что вскрытие со связанными ключами для выбранных открытых текстов уменьшает сложность вскрытия грубой силой почти вчетверо. Это вскрытие использует слабость использования ключа и может быть также применено, если алгоритм используется в кач естве однонаправленной хэш-функции (см. раздел 18.11).

Другое вскрытие со связанными ключами может вскрыть LOKI91 с помощью 2^{32} выбранных открытых текстов для выбранных ключей или с помощью 2^{48} известных открытых текстов для выбранных ключей [158]. Это вскрытие не зависит от числа этапов алгоритма. (В той же работе Бихам вскрывает LOKI89, используя кри птоанализ со связанными ключами, с помощью 2^{17} выбранных открытых текстов для выбранных ключей или с помощью 2^{33} известных открытых текстов для выбранных ключей.) Несложно повысить устойчивость LOKI91 к вскрытию такого типа, усложнив схему использования ключа.

Патенты и лицензии

LOKI не запатентован. Кто угодно может реализовать алгоритм и использовать его. Исходный код, прив еденный в этой книге, написан в Университете Нового Южного Уэльса. При желании использовать эту реализ ацию (или другие реализации, которые на несколько порядков быстрее) в коммерческом продукте обращайтесь к Директору CITRAD, Факультет компьютерных наук, Университетский колледж, Университет Нового Южного Уэльса, Академия австралийских вооруженных сил, Канберра, Австралия (Director CITRAD, Department of Computer Science, University College, UNSW, Australian Defense Force Academy, Canberra ACT 2600, Australia; FAX: +61 6 268 8581.

13.7 KHUFU и KHAFRE

В 1990 году Ральф Меркл (Ralph Merkle) предложил два алгоритма. В основе их проектирования лежали следующие принципы [1071]:

1. 56-битовый размер ключа DES слишком мал. Так как стоимость увеличения размера ключа прене брежимо мала (компьютерная память недорога и доступна), он должен быть увеличен.
2. Интенсивное использование перестановок в DES хотя и удобно для аппаратных реализаций, чрезв ычайно затрудняет программные реализации. Наиболее быстрые реализации DES выполняют перест ановки табличным образом. Просмотр таблицы может обеспечить те же характеристики "рассеяния", что и собственно перестано вки, и может сделать реализацию намного более гибкой.
3. S-блоки DES, всего с 64 4-битовыми элементами, слишком малы. Теперь с увеличением памяти должны увеличиться и S-блоки. Более того, все восемь S-блоков используются одновременно. Хотя это и удобно для аппаратуры, для программной реализации это кажется ненужным ограничением. Должны быть реализованы больший размер S-блоков и последовательное (а не параллельное) их и спользование.
4. Широко признано, что начальная и заключительная перестановки криптографически бессмысленны, поэтому они должны быть устранены.

5. Все быстрые реализации DES заранее рассчитывают ключи для каждого этапа. При данном условии нет смысла усложнять эти вычисления.

6. В отличие от DES критерии проектирования S-блоков должны быть общедоступны.

К этому перечню Меркл, возможно, теперь добавил бы "устойчивость к дифференциальному и линейному криптоанализу", ведь в то время эти способы вскрытия не были известны.

Khufu

Khufu - это 64-битовый блочный шифр. 64-битовый открытый тест сначала разбивается на две 32-битовые половины, *L* и *R*. Над обеими половинами и определенными частями ключа выполняется операция XOR. Затем, аналогично DES, результаты проходят через некоторую последовательность этапов. На каждом этапе младший значащий байт *L* используется в качестве входных данных S-блока. У каждого S-блока 8 входных битов и 32 выходных бита. Далее выбранный в S-блоке 32-битовый элемент подвергается операции XOR с *R*. Затем *L* циклически сдвигается не несколько из восьми битов, *L* и *R* меняются местами, и этап заканчивается. Сам S-блок не является статическим, но меняется каждые восемь этапов. Наконец после последнего этапа над *L* и *R* выполняется операция XOR с другими частями ключа, и половины объединяются, образуя блок шифротекста.

Хотя части ключа используются для XOR с блоком шифрования в начале и в конце алгоритма, главная цель ключа -генерация S-блоков. Эти S-блоки - секретны, по сути являются они являются частью ключа. Полный размер ключа Khufu равен 512 битам (64 байтам), алгоритм предоставляет способ генерации S-блоков по кл ючу. Количество этапов алгоритма остается открытым. Меркл упомянул, что 8-этапный Khufu чувствителен к вскрытию с выбранным открытым текстом и рекомендует 16, 24 или 32 этапа [1071]. (Он ограничивает выбор количества этапов числами, кратными восьми.)

Так как в Khufu используются зависимые от ключа и секретные S-блоки, он устойчив к дифференциальному криптоанализу. Существует дифференциальное вскрытие 16-этапного Khufu, которое раскрывает ключ после 2^{31} выбранных открытых текстов [611], но его не удалось расширить на большее количество этапов. Если лучшим способом вскрыть Khufu является грубая сила, то его надежность производит сильное впечатление. 512-битовый ключ обеспечивает сложность 2^{512} - огромное число при любых условиях.

Khafre

Khafre - это вторая из криптосистем, предложенных Мерклом [1071]. (Khufu (Хуфу) и Khafre (Хафр) - это имена египетских фараонов.) По конструкции этот алгоритм похож на Khufu, но он спроектирован для прил ожений, не использующих предварительных вычислений. S-блоки не зависят от ключа. Вместо этого Khafre и с пользует фиксированные S-блоки. Блок шифрования подвергается операции XOR с ключом не только перед первым этапом и после последнего, но и после каждых 8 этапов шифрования.

Меркл предположил, что с Khafre должны использоваться 64- или 128-битовые ключи, и что для Khafre п отребуется больше этапов, чем для Khufu. Это наряду с тем, что каждый этап Khafre сложнее этапа Khufu, делает Khafre более медленным. Зато для Khafre не нужны никакие предварительны расчеты, что позволяет быстрее шифровать небольшие порции данных.

В 1990 году Бихам и Шамир применили свой метод дифференциального анализа против Khafre [170]. Им удалось взломать 16-этапный Khafre с помощью вскрытия с выбранным открытым текстом после 1500 разли чных шифрований. На их персональном компьютере это заняло около часа. Преобразование этого вскрытия во вскрытие с известным открытым текстом потребует около 238 шифрований. Khafre с 24 этапами может быть вскрыт с помощью вскрытия с выбранным открытым текстом за 253 шифрования, а с помощью вскрытия с и звестным открытым текстом - за 259 шифрования.

Патенты

И Khufu, и Khafre запатентованы [1072]. Исходный код этих алгоритмов содержится в патенте. При желании получить лицензию на любой или оба алгоритма следует обратиться к директору по лицензированию корпор ации Xerox (Director of Licensing, Xerox Corporation, P.0. Box 1600, Stamford, CT, 06904-1600).

13.8 RC2

RC2 представляет собой алгоритм с переменной длиной ключа, спроектированный Роном Ривестом (Ron Rivest) для RSA Data Security, Inc. (RSADSI). Очевидно "RC" - это сокращенное "Ron's Code" ("Код Рона"), хотя официально это "Rivest Cipher" ("Шифр Ривеста"). (RC3 был взломан в RSADSI в процессе разработки, RC1 не вышел за пределы записной книжки Ривеста.) Он представляет собой частную собственность, и его детали не были опубликованы. Не думайте ни минуты, что это увеличивает его безопасность. RC2 уже появился в ко ммерческих продуктах. Насколько мне известно, RC2 не был запатентован и защищен только как торговый се крет.

RC2 - это шифр с 64-битовым блоком и переменной длиной ключа, предназначенный заменить DES. В соо тветствии с утверждениями компании программные реализации RC2 в три раза быстрее DES. Алгоритм может использовать ключ переменной длины, от 0 байтов до максимальной длины строки, поддерживаемой компь ютерной системой, скорость шифрования не зависит от размера ключа. Этот ключ предварительно используется для заполнения 128-байтовой таблицы, зависящей от ключа. Поэтому множество действительно различных ключей составляет 21024. RC2 не использует S-блоков [805], используются две операции - "смешивание" и "перемешивание" ("mix" и "mash"), для каждого этапа выбирается одна из них. В соответствии с литературой [1334]:

> . . . RC2 не является итеративным блочным шифром. Это предполагает, что RC2 более устойчив к дифференциальному и линейному криптоанализу, чем другие блочные шифры, безопасность которых опирается на копирование схемы DES.

Отказ RSADSI опубликовать RC2 заставляет сомневаться в намерениях этой компании. Она обещает пр едоставить детали алгоритма всем, кто подпишет соглашение о нераспространении информации, и утверждает, что позволит криптоаналитикам опубликовать любые обнаруженные негативные результаты. Мне неизвестно ни об одном криптоаналитике, не работающем в этой компании, кто бы исследовал алгоритм, так как это по сути означало бы выполнить работу по анализу для компании.

Тем не менее, Рон Ривест - не шарлатан. Он уважаемый и компетентный криптограф. Я лично в значител ьной степени верю в этот алгоритм, хотя я лично и не видел кода. RC4, также являющийся интеллектуальной собственностью RSADSI, был опубликован в Internet (см. раздел 17.1), и, вероятно, опубликование RC2 являе тся только вопросом времени.

По соглашению между Ассоциацией издателей программного обеспечения (Software Publishers Association, SPA) и правительством США RC2 и RC4 (см. раздел 17.1) получили специальный экспортный статус (см. ра здел 25.14). Процесс получения разрешения на экспорт продуктов, реализующих один из этих двух алгоритмов, значительно упрощен при условии, что длина ключа не превышает 40 битов.

Достаточен ли 40-битовый ключ? Существует всего один триллион возможных ключей. При условии, что наиболее эффективным методом криптоанализа является вскрытие грубой силой (большое допущение, ведь а лгоритм никогда не был опубликован), и что микросхема грубого вскрытия может проверить миллион ключей в секунду, поиск правильного ключа займет 12.7 дней. Тысяча машин, работающих параллельно, смогут ра скрыть ключ за двадцать минут.

RSA Data Security, Inc., утверждает, что, хотя шифрование и дешифрирования выполняются для быстро, и счерпывающего поиска потребуется намного больше времени. Заметное количество времени тратится на форм ирование плана использования ключа. Хотя это время пренебрежимо мало при шифровании и дешифрировании сообщений, это не так при проверке каждого возможного ключа.

Правительство США никогда не позволило бы экспортировать любой алгоритм, который оно, по крайней мере в теории, не смогло бы вскрыть. Оно может создать магнитную ленту или CD с конкретным блоком о ткрытого текста, зашифрованным каждым возможным ключом. Для вскрытия сообщения остается только вст авить ленту и сравнить блоки шифротекста в сообщении с блоками шифротекста на ленте. При совпадении мо жно проверить возможный ключ и посмотреть, имеет ли сообщение какой-нибудь смысл. Если они выберут часто встречающийся блок (все нули, ASCII-символы пробела, и т.д.), этот метод будет работать. Объем данных, нужный для хранения результатов шифрования 64-битового блока открытого текста всеми 10^{12} возможными ключами, составляет 8 терабайтов - вполне реально. По поводу лицензирования RC2 обращайтесь в RSADSI (см. раздел 25.4).

13.9 IDEA

Первый вариант шифра IDEA, предложенный Ксуеджа Лай (Xuejia Lai) и Джеймсом Масси (James Massey), появился в 1990 году [929]. Он назывался PES (Proposed Encryption Standard, предложенный стандарт шифр ования). В следующем году, после демонстрации Бихамом и Шамиром возможностей дифференциального кри птоанализа, авторы усилили свой шифр против такого вскрытия и назвали новый алгоритм IPES (Improved Proposed Encryption Standard, улучшенный предложенный стандарт шифрования) [931, 924]. В 1992 году назв ание IPES было изменено на IDEA (International Data Encryption Algorithm, международный алгоритм шифров ания данных) [925].

IDEA основывается на некоторых впечатляющих теоретических положениях и, хотя криптоанализ добился некоторых успехов в отношении вариантов с уменьшенным количеством этапов, алгоритм все еще кажется сильным. По моему мнению это самый лучший и самый безопасный блочный алгоритм, опубликованный сег одня.

Будущее IDEA пока неясно. Попыток заменить им DES предпринято не было, частично потому, что он зап атентован и должен быть лицензирован для коммерческих приложений, и частично потому, что люди пока все еще ждут, наблюдая насколько хорошо поведет себя алгоритм в предстоящие годы криптоанализа. Его сег о-

дняшняя известность объясняется тем, что он является частью PGP (см. раздел 24.12).

Обзор IDEA

IDEA является блочным шифром, он работает с 64-битовыми блоками открытого текста. Длина ключа - 128 битов. Для шифрования и дешифрирования используется один и тот же алгоритм.

Как и другие, уже рассмотренные блочные шифры IDEA использует и запутывание, и рассеяние. Флософия, лежащая в основе проекта, представляет собой "объединение операций из различных алгебраических групп". Смешиваются три алгебраические группы, и все они могут быть легко реализованы как аппаратно, так и пр ограммно:

— XOR

— Сложение по модулю 2^{16}

— Умножение по модулю $2^{16} + 1$. (Это операцию можно рассматривать как S-блок IDEA.)

Все эти операции (а в алгоритме используются только они, перестановки на битовом уровне не применяю тся) работают с 16-битовыми подблоками. Этот алгоритм даже эффективнее на 16-битовых процессорах.

Описание IDEA

Схема IDEA представлена на Рис. 13-9. 64-битовый блок данных делится на четыре 16-битовых подблока: X_1, X_2, X_3 и X_4. Эти четыре подблока становятся входными данными для первого этапа алгоритма. Всего в алг оритме восемь этапов. На каждом этапе четыре подблока подвергаются операциям XOR, сложениям и умнож ениям друг с другом и с шестью 16-битовыми подключами. Между этапами обмениваются местами второй и третий подблоки. Наконец четыре подблока объединяются с четырьмя подключами в окончательном преобраз овании. На каждом этапе события происходят в следующей последовательности:

(1) Перемножаются X_1 и первый подключ.

(2) Складываются X_2 и второй подключ.

(3) Складываются X_3 и третий подключ.

(4) Перемножаются X_4 и четвертый подключ.

(5) Выполняется XOR над результатами этапов (1) и (3).

(6) Выполняется XOR над результатами этапов (2) и (4).

(7) Перемножаются результаты этапа (5) и пятый подключ.

(8) Складываются результаты этапов (6) и (7).

(9) Перемножаются результаты этапа (8) и шестой подключ.

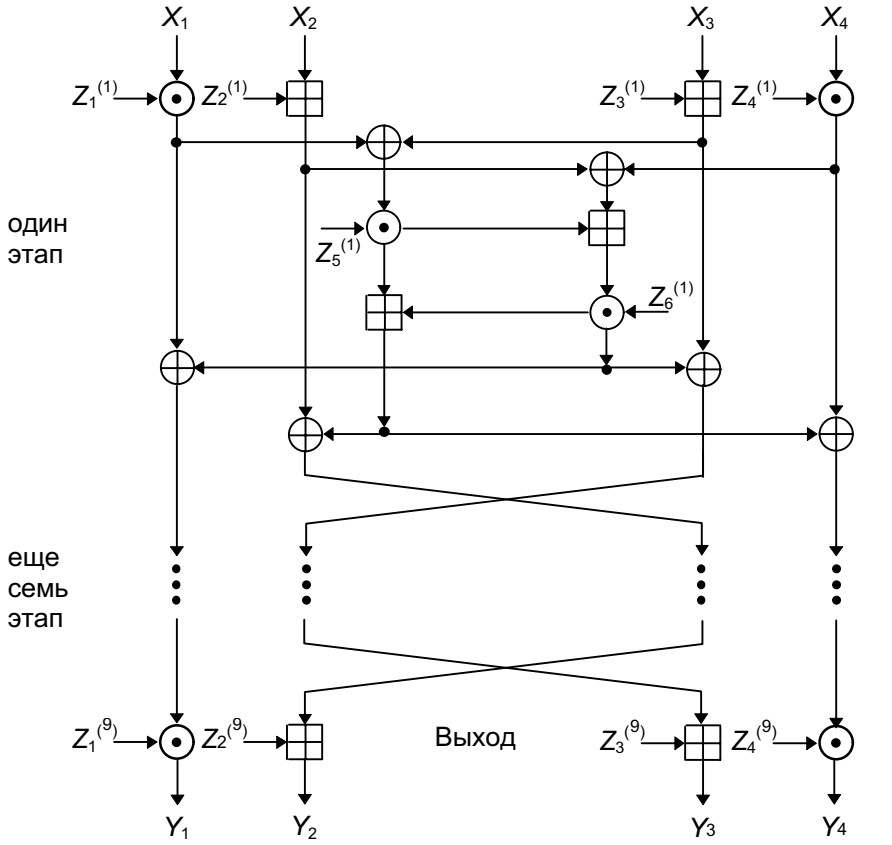
(10) Складываются результаты этапов (7) и (9).

(11) Выполняется XOR над результатами этапов (1) и (9).

(12) Выполняется XOR над результатами этапов (3) и (9).

(13) Выполняется XOR над результатами этапов (1) и (10).

(14) Выполняется XOR над результатами этапов (4) и (10).



X_i : 16-битовый подблок открытого текста
Y_i : 16-битовый подблок шифротекста
$Z_i^{(r)}$: 16-битовый подблок ключа
$\oplus$: побитовое "исключающее или" (XOR) 16-битовых подблоков
$\boxplus$: сложение по модулю 2^{16} 16-битовых целых
$\odot$: умножение по модулю $2^{16}+1$ 16-битовых целых при условии, что нулевой подблок соответствует 2^{16}

Рис. 13-9. IDEA.

Выходом этапа являются четыре подблока - результаты действий (11), (12), (13) и (14). Поменяйте местами два внутренних подблока (но не в последнем этапе), и вы получите исходные данные для следующего этапа.

После восьмого этапа выполняется заключительное преобразование:

(1) Перемножаются X_1 и первый подключ.

(2) Складываются X_2 и второй подключ.

(3) Складываются X_3 и третий подключ.

(4) Перемножаются X_4 и четвертый подключ.

Наконец четыре подблока снова соединяются, образуя шифротекст.

Также несложно создавать подключи. Алгоритм использует 52 из них (шесть для каждого из восьми этапов и еще четыре для заключительного преобразования). Сначала 128-битовый ключ делится на восемь 16-битовых подключей. Это первые восемь подключей алгоритма (шесть для первого этапа и два - для второго). Затем ключ циклически сдвигается налево на 25 битов и снова делится на восемь подключей. Первые четыре используются на этапе 2, а оставшиеся четыре - на этапе 3. Ключ циклически сдвигается налево на 25 битов для получения следующих восьми подключей, и так до конца алгоритма.

Дешифрирование выполняется точно также за исключением того, что подключи инвертируются и слегка и з-меняются. Подключи при дешифрировании представляют собой обратные значения ключей шифрования по отношению к операциям либо сложения, либо умножения. (Для IDEA подблоки, состоящие из одних нулей, считаются равными $2^{16} = -1$ для умножения по модулю $2^{16} + 1$, следовательно, обратным значением 0 относ и-тельно умножения является 0.) Эти вычисления могут занять некоторое время, но их нужно выполнить один раз для каждого ключа дешифрирования. В Табл. 13-4 представлены подключи шифрования и соответствующие

подключи дешифрирования.

Табл. 13-4.
Подключи шифрования и дешифрирования IDEA

Этап	Подключи шифрования						Подключи дешифрирования					
1	$Z_1^{(1)}$	$Z_2^{(1)}$	$Z_3^{(1)}$	$Z_4^{(1)}$	$Z_5^{(1)}$	$Z_6^{(1)}$	$Z_1^{(9)-1}$	$-Z_2^{(9)}$	$-Z_3^{(9)}$	$Z_4^{(9)-1}$	$Z_5^{(8)}$	$Z_6^{(8)}$
2	$Z_1^{(2)}$	$Z_2^{(2)}$	$Z_3^{(2)}$	$Z_4^{(2)}$	$Z_5^{(2)}$	$Z_6^{(2)}$	$Z_1^{(8)-1}$	$-Z_2^{(8)}$	$-Z_3^{(8)}$	$Z_4^{(8)-1}$	$Z_5^{(7)}$	$Z_6^{(7)}$
3	$Z_1^{(3)}$	$Z_2^{(3)}$	$Z_3^{(3)}$	$Z_4^{(3)}$	$Z_5^{(3)}$	$Z_6^{(3)}$	$Z_1^{(7)-1}$	$-Z_2^{(7)}$	$-Z_3^{(7)}$	$Z_4^{(7)-1}$	$Z_5^{(6)}$	$Z_6^{(6)}$
4	$Z_1^{(4)}$	$Z_2^{(4)}$	$Z_3^{(4)}$	$Z_4^{(4)}$	$Z_5^{(4)}$	$Z_6^{(4)}$	$Z_1^{(6)-1}$	$-Z_2^{(6)}$	$-Z_3^{(6)}$	$Z_4^{(6)-1}$	$Z_5^{(5)}$	$Z_6^{(5)}$
5	$Z_1^{(5)}$	$Z_2^{(5)}$	$Z_3^{(5)}$	$Z_4^{(5)}$	$Z_5^{(5)}$	$Z_6^{(5)}$	$Z_1^{(5)-1}$	$-Z_2^{(5)}$	$-Z_3^{(5)}$	$Z_4^{(5)-1}$	$Z_5^{(4)}$	$Z_6^{(4)}$
6	$Z_1^{(6)}$	$Z_2^{(6)}$	$Z_3^{(6)}$	$Z_4^{(6)}$	$Z_5^{(6)}$	$Z_6^{(6)}$	$Z_1^{(4)-1}$	$-Z_2^{(4)}$	$-Z_3^{(4)}$	$Z_4^{(4)-1}$	$Z_5^{(3)}$	$Z_6^{(3)}$
7	$Z_1^{(7)}$	$Z_2^{(7)}$	$Z_3^{(7)}$	$Z_4^{(7)}$	$Z_5^{(7)}$	$Z_6^{(7)}$	$Z_1^{(3)-1}$	$-Z_2^{(3)}$	$-Z_3^{(3)}$	$Z_4^{(3)-1}$	$Z_5^{(2)}$	$Z_6^{(2)}$
8	$Z_1^{(8)}$	$Z_2^{(8)}$	$Z_3^{(8)}$	$Z_4^{(8)}$	$Z_5^{(8)}$	$Z_6^{(8)}$	$Z_1^{(2)-1}$	$-Z_2^{(2)}$	$-Z_3^{(2)}$	$Z_4^{(2)-1}$	$Z_5^{(1)}$	$Z_6^{(1)}$
заключительное преобразование	$Z_1^{(9)}$	$Z_2^{(9)}$	$Z_3^{(9)}$	$Z_4^{(9)}$			$Z_1^{(1)-1}$	$-Z_2^{(1)}$	$-Z_3^{(1)}$	$Z_4^{(1)-1}$		

Скорость IDEA

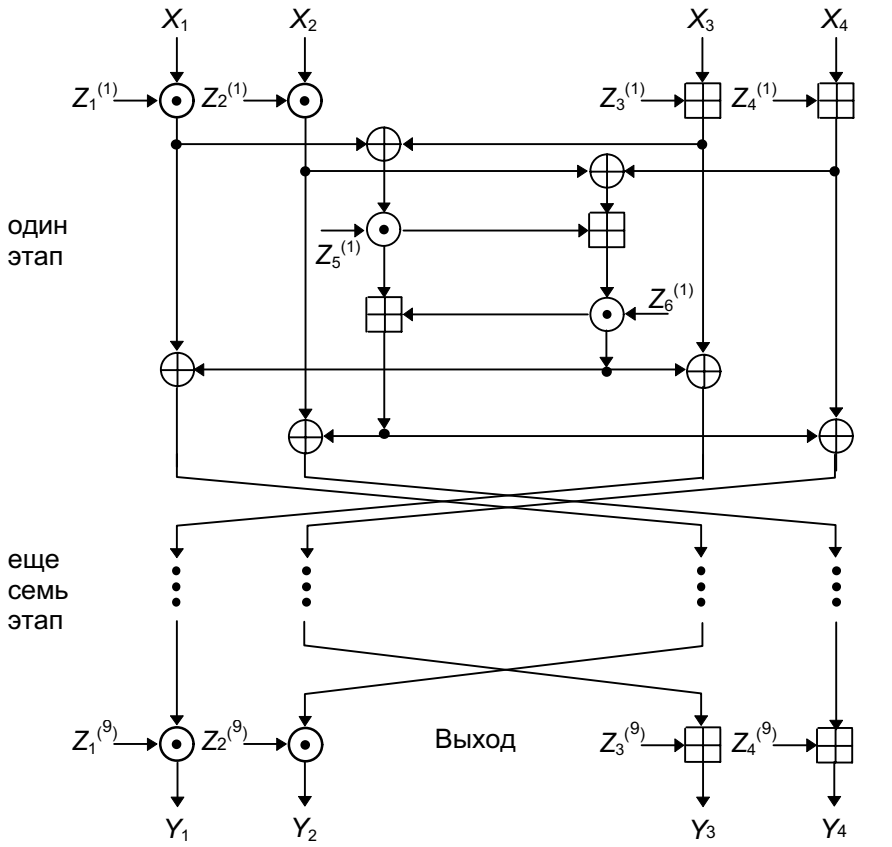
Современные программные реализации IDEA примерно в два раза быстрее, чем DES. На компьютере с i386/33 МГц IDEA шифрует данные со скоростью 880 Кбит/с, а на компьютере с i486/33 МГц - со скоростью 2400 Кбит/с. Вы могли подумать, что IDEA должен был быть побыстрее, но умножения - недешевое удовольс т-вие. Умножение двух 32-битовых чисел на процессоре i486 занимает 40 тактов (10 на процессоре Pentium).

Реализация PES на базе СБИС шифрует данные со скоростью 55 Мбит/с при тактовой частоте 25 МГц [208,398]. Другая СБИС, разработанная ETH Zurich и состоящая из of 251000 транзисторов на кристалле пл о-щадью 107.8 мм^2, шифрует данные с помощью алгоритма IDEA со скоростью 177 Мбит/с при тактовой частоте 25 МГц [926, 207, 397].

Криптоанализ IDEA

Длина ключа IDEA равна 128 битам - более чем в два раза длиннее ключа DES. При условии, что наиболее эффективным является вскрытие грубой силой, для вскрытия ключа потребуется 2^{128} (10^{38}) шифрований. Созд айте микросхему, которая может проверять миллиард ключей в секунду, объедините миллиард таких микр о-схем, и вам потребуется 10^{13} лет для решения проблемы - это больше, чем возраст вселенной. 10^{24} таких микро-схем могут найти ключ за день, но во вселенной не найдется столько атомов кремния, чтобы построить такую машину. Наконец мы чего-то достигли, хотя в некоторых темных вопросах я лучше останусь сторонним набл ю-дателем.

Может быть вскрытие грубой силой - не лучший способ вскрытия IDEA. Алгоритм все еще слишком нов, чтобы можно было говорить о каких-то конкретных криптографических результатах. Разработчики сделали все возможное, чтобы сделать алгоритм устойчивым к дифференциальному криптоанализу. Они определили пон я-тие марковского шифра и продемонстрировали, что устойчивость к дифференциальному криптоанализу может быть промоделирована и оценена количественно [931, 925]. (Для сравнения с алгоритмом IDEA, устойчивость которого к дифференциальному криптоанализу была усилена, и который показан на Рис. 13-9, на Рис. 13-10 приведен первоначальный алгоритм PES. Удивительно, как такие незначительные изменения могут привести к столь большим различиям.) В [925] Лай (Lai) утверждал (он привел подтверждение, но не доказательство), что IDEA устойчив к дифференциальному криптоанализ уже после 4 из 8 этапов. Согласно Бихаму, его попытка вскрыть IDEA с помощью криптоанализа со связанными кл ючами также не увенчалась успехом [160].



X_i : 16-битовый подблок открытого текста
Y_i : 16-битовый подблок шифротекста
$Z_i^{(r)}$: 16-битовый подблок ключа
$\oplus$: побитовое "исключающее или" (XOR) 16-битовых подблоков
$\boxplus$: сложение по модулю 2^{16} 16-битовых целых
$\odot$: умножение по модулю $2^{16}+1$ 16-битовых целых при условии, что нулевой подблок соответствует 2^{16}

Рис. 13-10. PES.

Вилли Майер (Willi Meier) исследовал три алгебраических операции IDEA и показал, что, хотя они нес о-вместимы, есть случаи, когда эти операции можно упростить так, чтобы в некоторой степени облегчить [1050]. Его вскрытие 2-этапного IDEA оказалось эффективнее вскрытия грубой силой (2^{42} операций), но для IDEA с 3 и более этапами эффективность этого вскрытия была ниже вскрытия грубой силой. Безопасность полного 8-этапного IDEA осталась непоколебимой.

Джоан Дэймен (Joan Daemen) открыла класс слабых ключей IDEA [405, 409]. Эти ключи не являются сл а-быми в том смысле, в котором слабы некоторые ключи DES, для которых функция шифрования обратна самой себе. Слабость этих ключей состоит в том, что взломщик может легко определить их с помощью вскрытия с выбранным открытым текстом. Например, слабым является следующий ключ (в шестнадцатиричной записи):

0000,0000,0*x*00,0000,0000,000*x*,*xxxx*,*x*000

В позиции "*x*" может стоять любая цифра. При использовании такого ключа побитовое XOR определенных пар открытых текстов равно побитовому XOR получившихся пар шифротекстов.

В любом случае вероятность случайной генерации одного из таких слабых ключей очень мала: $1/2^{96}$. Опас-ность случайно выбрать такой ключ практически не существует. К тому же, несложно модифицировать IDEA так, чтобы исключить наличие слабых ключей - достаточно выполнить XOR каждого подключа с числом 0x0dae [409].

Хотя попыток выполнить криптоанализ IDEA было много, мне неизвестно ни об одной успешной.

Режимы работы и варианты IDEA

IDEA может работать в любом из режимов работы блочного шифра, описанных в главе 9. Против двойных реализаций IDEA может быть предпринято то же вскрытие "встреча посередине", что и против DES (см. раздел

15.1). Однако, так как ключ IDEA более чем в два раза длиннее ключа DES, это вскрытие непрактично. Объем нужной для такого вскрытия памяти составит $64*2^{128}$ битов, или 10^{39} байтов. Может быть во вселенной и достаточно материи, чтобы построить такое хранилище, но я в этом сомневаюсь.

Если вы учитываете возможность использования параллельной вселенной, используйте утроенную реализацию IDEA (см. раздел 15.2):

$$C = E_{K_3}(D_{K_2}(E_{K_1}(P)))$$

Такая реализация устойчива против вскрытия "встреча посередине".

Кроме того, почему бы вам не реализовать IDEA независимыми подключами, особенно если ваши средства распределения ключей позволяют работать с длинными ключами. Для IDEA нужно всего 52 16-битовых ключа, общей длиной 832 битов. Этот вариант определенно безопасней, но никто не сможет сказать насколько.

В наивной модификации может быть увеличен вдвое размер блока. Алгоритм также прекрасно работал бы с 32-битовыми подблоками вместо 16-битовых и с 256-битовым ключом. Шифрование выполнялось бы быстрее, и безопасность возросла бы в 2^{32} раза. Или нет? Теория, на которой основан алгоритм, опирается на то, что $2^{16}+1$ является простым числом. А $2^{32} + 1$ простым числом не является. Может быть алгоритм и можно изменить так, чтобы он работал, но его безопасность будет совсем иной. Лай говорит, что заставить работать такой алгоритм будет нелегко [926].

Хотя IDEA кажется намного безопаснее DES, не всегда можно легко заменить один алгоритм другим в существующем приложении. Если ваша база данных и шаблоны сообщений могут работать с 64-битовым ключом, реализация 128-битового ключа IDEA может быть возможной.

Для таких приложений создайте 128-битовый ключ, объединив 64-битовый ключ сам с собой. Не забывайте, что эта модификация заметно ослабляет IDEA.

Если вас больше волнует скорость работы, а не безопасность, попробуйте вариант IDEA с меньшим числом этапов. Сегодня лучшее вскрытие IDEA быстрее вскрытия грубой силой только для 2.5 и менее этапов [1050], 4-этапный IDEA будет в два раза быстрее и, насколько мне известно, его безопасность не уменьшится.

Caveat Emptor[1]

IDEA - это относительно новый алгоритм, многие вопросы пока остаются открытыми. Образует ли IDEA группу? (Лай думает, что нет [926].) Не существует ли пока не открытых способов вскрытия этого шифра? У IDEA твердая теоретическая основа, но снова и снова казавшиеся безопасными алгоритмы капитулируют перед новыми формами криптоанализа. Ряд групп академических и военных исследователей не опубликовали свои результаты криптоанализа IDEA. Возможно, кто-нибудь уже добился или когда-нибудь добьется успеха.

Патенты и лицензии

IDEA запатентован в Европе и Соединенных Штатах [1012, 1013]. Патент принадлежит Ascom-Tech AG. Для некоммерческого использования лицензирование не нужно. При заинтересованности в лицензии для коммерческого применения алгоритма следует обратиться по адресу Ascom Systec AG, Dept CMVV, Cewerbepark, CH-5506, Mgenwil, Switzerland; +41 64 56 59 83; Fax: +41 64 56 59 90; idea@ascom.ch.

13.10 MMB

Недовольство использованием в IDEA 64-битового блока шифрования привело к созданию Джоном Дэймоном алгоритма под названием MMB (Modular Multiplication-based Block cipher, модульный блочный шифр, использующий умножения) [385, 405, 406]. В основе MMB лежит теория, используемая и в IDEA: перемешивающие операции из различных групп. MMB - это итеративный алгоритм, главным образом состоящий из линейных действий (XOR и использование ключа) и параллельное использование четырех больших нелинейных и зменяющих обычный порядок подстановок. Эти подстановки определяются с помощью умножения по модулю 2^{32}-1 с постоянными множителями. Результатом применения этих действий является алгоритм, использующий и 128-битовый ключ и 128-битовый блок.

MMB оперирует 32-битовыми подблоками текста (x_0, x_1, x_2, x_3) и 32-битовыми подблоками ключа (k_0, k_1, k_2, k_3). Это делает удобным реализацию алгоритма на современных 32-битовых процессорах. Чередуясь с XOR, шесть раз используется нелинейная функция f. Вот этот алгоритм (все операции с индексами выполняются по модулю 3):

$x_i = x_i \oplus k_i$, для $i = 0$ до 3

---

[1] Предупреждение покупателю

$f(x_0, x_1, x_2, x_3)$

$x_i = x_i \oplus k_{i+1}$, для $i = 0$ до 3

$f(x_0, x_1, x_2, x_3)$

$x_i = x_i \oplus k_{i+2}$, для $i = 0$ до 3

$f(x_0, x_1, x_2, x_3)$

$x_i = x_i \oplus k_i$, для $i = 0$ до 3

$f(x_0, x_1, x_2, x_3)$

$x_i = x_i \oplus k_{i+1}$, для $i = 0$ до 3

$f(x_0, x_1, x_2, x_3)$

$x_i = x_i \oplus k_{i+2}$, для $i = 0$ до 3

$f(x_0, x_1, x_2, x_3)$

У функции f три этапа:

(1) $x_1 = c_i * x_i$, для $i = 0$ до 3 (Если на входе умножения одни единицы, то на выходе - тоже одни единицы.)

(2) Если младший значащий бит $x_0 = 1$, то $x_0 = x_0 \oplus C$. Если младший значащий бит $x_3 = 0$, то $x_3 = x_3 \oplus C$.

(3) $x_i = x_{i-1} \oplus x_i \oplus x_{i+1}$, для $i = 0$ до 3

Все операции с индексами выполняются по модулю 3. Операция умножения на этапе (1) выполняется по м одулю 2^{32}-1. В данном алгоритме если второй операнд - это 2^{32}-1, то результат также равен 2^{32}-1. В алгоритме используются следующие константы:

C = 2aaaaaaa

c_0 = 025f1cdb

$c_1 = 2 * c_0$

$c_2 = 2^3 * c_0$

$c_3 = 2^7 * c_0$

Константа C - это "простейшая" константа с высоким троичным весом, нулевым младшим значащим битом и без круговой симметрии. У константы c0 несколько иные характеристики. Константы c_1, c_2 и c_3 являются смещенными версиями c_0, и используются для предотвращения вскрытий основанных на симметрии. Подробности можно найти в [405].

Дешифрирование является обратным процессом. Этапы (2) и (3) заменяются на свою инверсию. На этапе (1) вместо c_i^{-1} используется c_i. c_i^{-1} = 0dad4694.

Безопасность MMB

Схема MMB обеспечивает на каждом этапе значительное и независимое от ключа рассеяние. В IDEA ра ссеяние до определенной степени зависит от конкретных подключей. В отличие от IDEA у MMB нет слабых ключей.

К сожалению MMB - это умерший алгоритм [402]. Это утверждение справедливо по многим причинам, хотя криптоанализ MMB и не был опубликован. Во первых, он проектировался без учета требований устойчивости к линейному криптоанализу. Выбор мультипликативных множителей обеспечил устойчивость к дифференциал ьному криптоанализу, но о линейном криптоанализе авторам алгоритма было еще неизвестно.

Во вторых, Эли Бихам реализовал эффективное вскрытие с выбранным ключом [160], использующеее тот факт, что все этапы идентичны, а ключ при использовании просто циклически сдвигается на 32 бита. В третьих, несмотря на то, что программные реализации MMB были бы очень эффективны, в аппаратном исполнении а лгоритм менее эффективен, чем DES.

Дэймон предлагает, что тот, кто захочет улучшить MMB, должен сначала проанализировать умножение по модулю с помощью линейного криптоанализа и подобрать новый множитель, а затем сделать константу C ра зличной для каждого этапа [402]. Затем, улучшив использование ключа, добавляя к ключам этапов константы с целью устранения смещения. Но сам не стал заниматься этим и разработал 3-Way (см. раздел 14.5).

13.11 CA-1.1

CA - это блочный шифр, основанный на клеточных автоматах и разработанный Говардом Гутовицом (Howard Gutowitz) [677, 678, 679]. Он шифрует 384-битовые блоки открытого текста 1088-битовым ключом (на самом деле используется два ключа - 1024-битовый и 64- битовый). Из-за природы клеточных автоматов алг оритм наиболее эффективен при реализации в больших параллельных интегрированных схемах.

CA-1.1 использует как обратимые, так и необратимые правила клеточного автомата. При обратимом прав иле каждое состояние структуры получается из единственного предшествующего состояния, а при необратимом правиле у каждого состояния может быть несколько предшественников. При шифровании необратимые правила пошагово обращаются во времени. Для продвижения обратно от текущего состояния случайным образом дол жно выбираться одно из состояний-предшественников. Этот процесс многократно повторяется. Таким образом, обратная итерация служит для смешивания случайной информации с инфорамацией сообщения. CA-1.1 испол ьзует особый сорт частично линейного необратимого правила, такого, что для любого данного состояния может быть быстро построено случайное состояние-предшественник. На некоторых стадиях шифрования используются и обратимые правила.

Обратимые правила (простые параллельные перестановки подблоков состояния) нелинейны. Необратимые правила полностью определяются ключом, а обратимые зависят как от ключа, так и от случайной информации, вставленной в ходе шифрования необратимыми правилами.

CA-1.1 основан на структуре блочных связей. То есть, обработка блока сообщения частично отделена от о бработки потока случайной информации, вставленной при шифровании. Эта случайная информация служит для связи друг с другом стадий шифрования. Она также может быть использована для связи с потоком шифроте кста. Информация связи генерируется как часть шифрования.

Так как CA-1.1 представляет собой новый алгоритм, слишком рано делать какие-либо заявления о его без опасности. Гутовиц упоминает некоторые возможные вскрытия, включая дифференциальный криптоанализ, но ему не удалось вскрыть алгоритм. В качестве стимула Гутовиц предложил награду в 1000 долларов для "первого человека, который разработает доступную процедуру вскрытия CA-1.1."

CA-l.1 запатентован [678], но доступен для некоммерческого использования. При необходимости получить лицензию на алгоритм или объявленную награду за криптоанализ обращайтесь к Говарду Гутовицу по адресу Howard Cutowitz, ESPCI, Laboratorie d'Electronique, 10 rue Vauquelin, 75005 Paris, France.

13.12 SKIPJACK

Skipjack разработан NSA в качестве алгоритма шифрования для микросхем Clipper и Capstone (см. разделы 24.16 и 24.17). Так как этот алгоритм объявлен секретным, его подробности никогда не публиковались. Он б удет реализован только как защищенная от взлома аппаратура.

Этот алгоритм объявлен секретным не потому, что это повышает его надежность, а потому что NSA не х очет, чтобы Skipjack использовался без механизма условного вручения ключей Clipper. Агентство не хочет, чт обы программные реализации алгоритма распространились по всему миру.

Безопасен ли Skipjack? Если NSA захочет создать безопасный алгоритм, оно, скорее всего, это сделает. С другой стороны, если NSA захочет создать алгоритм с лазейкой, то оно сможет сделать и это. Вот что было опубликовано [1154, 462].

— Это итеративный блочный шифр.

— Размер блока - 64 бита.

— Алгоритм использует 80-битовый ключ.

— Он может быть использован в режимах ECB, CBC, 64-битовый OFB, либо 1-, 8-, 16-, 32- или 64-битовый CFB.

— Операция шифрования или дешифрирования состоит из 32 этапов.

— NSA начало работу над ним в 1985 и завершило проверку в 1990.

В документации на микросхему Mykotronx Clipper утверждается, что задержка в выдаче результата, прис ущая алгоритму Skipjack, составляет 64 такта. Это означает, что на каждый этап приходится два такта: один предположительно для подстановки с помощью S-блока, а другой - для заключительного XOR в конце каждого этапа. (Не забывайте, перестановки при аппаратных реализациях не занимают времени.) В документации Mykotronx эта двухтактная операция называется "G-блоком", а все вместе - "сдвигом". (Часть G-блока носит название "F-таблицы" и является таблицей констант, а может быть таблицей функций.)

По одним слухам Skipjack использует 16 S-блоков, а по другим для хранения S-блоков нужно всего 128 байт

памяти. Непохоже, чтобы оба этих слуха были правдой.

Еще один слух утверждает, что этапы Skipjack, в отличие от DES, работают не с половиной блока. Это вместе с замечанием о "сдвигах" и случайном заявлении на Crypto '94 о том, что в Skipjack применяется "48-битовая внутренняя структура", позволяет сделать вывод, что алгоритм по своей схеме похож на SHA (см. раздел 18.7), но использует четыре 16-битовых подблока. Три подблока, обработанные зависящей от ключа однонаправленной функцией, дают 16 битов, которые подвергаются операции XOR с оставшимся подблоком. Затем весь блок циклически сдвигается на 16 битов и поступает на вход следующего этапа, или сдвига. При этом также используются 128 байтов данных S-блока. Я подозреваю, что S-блоки зависят от ключа.

По своей структуре Skipjack вероятно похож на DES. NSA понимает, что его защищенная от взлома аппаратура в конце концов будет вскрыта и исследована, они не будут рисковать никакими передовыми криптографическими методами.

То, что NSA планирует использовать алгоритм Skipjack для шифрования своей Системы защиты сообщений (Defense Messaging System, DMS), свидетельствует о безопасности алгоритма. Чтобы убедить скептиков, NIST разрешил комиссии "уважаемых неправительственных экспертов . . . получить доступ к конфиденциальным подробностям алгоритма, чтобы они исследовали его возможности и опубликовали результаты своих исследований " [812].

В предварительном отчете этой комиссии экспертов [262] (окончательного отчета не было, и возможно никогда не будет) сообщалось:

> Принимая во внимание, что стоимость вычислительных мощностей уменьшается в два раза каждые 18 месяцев, сложность вскрытия Skipjack сравняется с сегодняшней сложностью вскрытия DES только через 36 лет. Следовательно, риск, что Skipjack будет взломан в ближайшие 30-40 лет, незначителен.
>
> Незначителен и риск взлома Skipjack с помощью более быстрых способов вскрытия, включая дифференциальный криптоанализ. У алгоритма не слабых ключей, отсутствует и свойство комплиментарности. Эксперты в отсутствие времени для самостоятельного большого исследования алгоритма изучили представленное NSA описание разработки и проверки алгоритма
>
> Устойчивость Skipjack к криптоанализу не зависит от хранения в тайне самого алгоритма.

Итак, участники дискуссии не смогли поработать с алгоритмом достаточно долго, чтобы прийти к каким-нибудь выводам самостоятельно. Все, что они смогли сделать - это взглянуть на результаты, показанные им NSA.

Остался без ответа вопрос, является ли плоским пространство ключей Skipjack (см. раздел 8.2). Даже если у Skipjack нет ключей, слабых в смысле DES, ряд особенностей процесса использования ключа может сделать одни ключи сильнее других. У Skipjack может быть 2^{70} сильных ключей, гораздо больше чем у DES, вероятность случайно выбрать один из этих сильных ключей будет приблизительно 1 к 1000. Лично я думаю, что пространство ключей Skipjack - плоское, но то, что об этом никто не заявил публично, вызывает тревогу.

Skipjack запатентован, но в соответствии с соглашением о секретности патента [1122] этот патент хранится в тайне. Патент будет опубликован тогда и только тогда, когда алгоритм Skipjack будет успешно восстановлен кем-то посторонним. Это дает возможность правительству воспользоваться и преимуществом защиты патентом, и преимуществом конфеденциальности торгового секрета.

Глава 14
И еще о блочных шифрах

14.1 ГОСТ

ГОСТ - это блочный алгоритм, разработанный в бывшем Советском Союзе [655, 1393]. Название "ГОСТ" является сокращением от "Государственный стандарт", нечто похожее на FIPS за исключением того, что это название могут носить стандарты практически любого типа. (Полным названием является "Государственный стандарт Союза ССР", или "Государственный стандарт Союза Советских Социалистических Республик".) Н омер данного стандарта - 28147-89. Все эти стандарты утверждаются Государственным комитетом по стандартам Союза ССР.

Я не знаю, использовался ли ГОСТ 28147-89 для засекреченного трафика или только для гражданского шифрования. Замечание в начале стандарта гласит, что алгоритм "удовлетворяет всем криптографическим тр е-бованиям, а степень защищаемой информации не ограничивается". Я слышал утверждения, что этот алгоритм первоначально использовался только для очень важных линий связи, включая секретные военные коммуник а-ции, но у меня нет подтверждений.

Описание ГОСТ

ГОСТ является 64-битовым алгоритмом с 256-битовым ключом. ГОСТ также использует дополнительный ключ, который рассматривается ниже. В процессе работы алгоритма на 32 этапах последовательно выполняется простой алгоритм шифрования.

Для шифрования текст сначала разбивается на левую половину L и правую половину R. На этапе i использу-ется подключ K_i. На этапе i алгоритма ГОСТ выполняется следующее:

$L_i = R_{i-1}$

$R_i = L_{i-1} \oplus \mathrm{f}(R_{i-1}, K_i)$

Этап ГОСТ показан на Рис. 14-1. Функция f проста. Сначала правая половина и i-ый подключ складываются по модулю 2^{32}. Результат разбивается на восемь 4-битовых кусочков, каждый из которых поступает на вход св о-его S-блока. ГОСТ использует восемь различных S-блоков, первые 4 бита попадают в первый S-блок, вторые 4 4 бита - во второй S-блок, и т.д. Каждый S-блок представляет собой перестановку чисел от 0 до 15. Например, S-блок может выглядеть как:

7, 10, 2, 4, 15, 9, 0, 3, 6, 12, 5, 13, 1, 8, 11

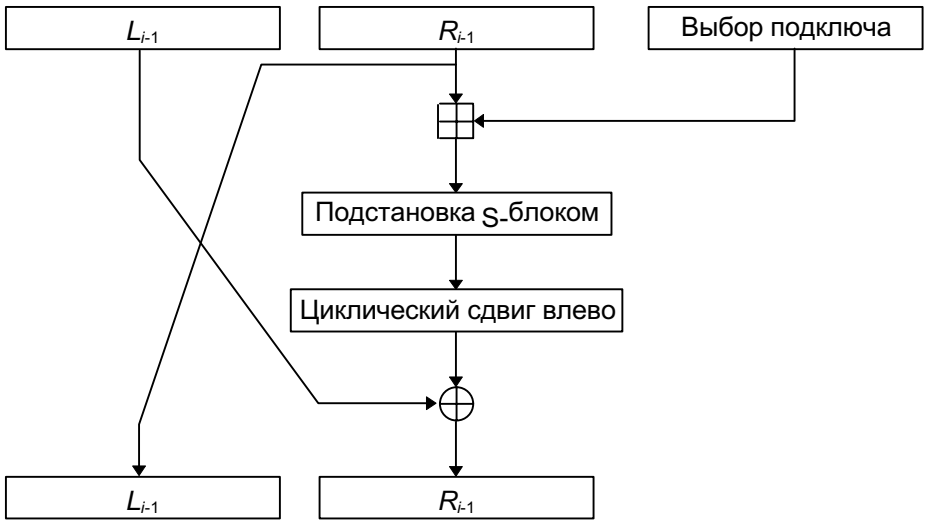


Рис. 14-1. Этап ГОСТ.

В этом случае, если на входе S-блока 0, то на выходе 7. Если на входе 1, на выходе 10, и т.д. Все восемь S-блоков различны, они фактически являются дополнительным ключевым материалом. S-блоки должны хр а-ниться в секрете.

Выходы всех восьми S-блоков объединяются в 32-битовое слово, затем все слово циклически сдвигается влево на 11 битов. Наконец результат объединяется с помощью XOR с левой половиной, и получается новая правая половина, а правая половина становится новой левой половиной. Выполните это 32 раза, и все в поря д-

ке.

Генерация подключей проста. 256-битовый ключ разбивается на восемь 32-битовых блоков: k_1, k_2, . . .k_8. На каждом этапе используется свой подключ, как показано в Табл. 14-1. Дешифрирование выполняется также, как и шифрование, но инвертируется порядок подключей k_i.

Стандарт ГОСТ не определяет способ генерации S-блоков, говорится только, что блоки должны быть пр е-доставлены каким-то образом [655]. Это породило домыслы о том, что советский производитель может поста в-лять хорошие S-блоки "хорошим" организациям и плохие S-блоки тем организациям, которых производитель собирается надуть. Это вполне может быть так, но неофициальные переговоры с российским производителем микросхем ГОСТ выявили другую альтернативу. Производитель создает перестановки S-блока самостоятельно с помощью генератора случайных чисел.

Табл. 14-1.
Использование подключей на различных этапах ГОСТ

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

Совсем недавно стал известным набор S-блоков, используемых в приложениях Центрального Банка РФ. Эти S-блоки также используются в однонаправленной хэш-функции ГОСТ (см. раздел 18.11) [657]. Они перечисл е-ны в Табл. 14-2.

Криптоанализ ГОСТ

Вот главные различия между DES и COST.

— DES использует сложную процедуру для генерации подключей из ключей. В ГОСТ эта процедура очень проста.

— В DES 56-битовый ключ, а в ГОСТ - 256-битовый. Если добавить секретные перестановки S-блоков, то полный объем секретной информации ГОСТ составит примерно 610 битов.

— У S-блоков DES 6-битовые входы и 4-битовые выходы, а у S-блоков ГОСТ 4-битовые входы и выходы. В обоих алгоритмах используется по восемь S-блоков, но размер S-блока ГОСТ равен одной четвертой размера S-блока DES.

— В DES используются нерегулярные перестановки, названные P-блоком, а в ГОСТ используется 11-битовый циклический сдвиг влево.

— В DES 16 этапов, а в ГОСТ - 32.

Табл. 14-2.
S-блоки ГОСТ

S-блок 1:															
4	10	9	2	13	8	0	14	6	11	1	12	7	15	5	3
S-блок 2:															
14	11	4	12	6	13	15	10	2	3	8	1	0	7	5	9
S-блок 3:															
5	8	1	13	10	3	4	2	14	15	12	7	6	0	9	11
S-блок 4:															
7	13	10	1	0	8	9	15	14	4	6	12	11	2	5	3
S-блок 5:															
6	12	7	1	5	15	13	8	4	10	9	14	0	3	11	2
S-блок 6:															

4	11	10	0	7	2	1	13	3	6	8	5	9	12	15	14
S-блок 7:															
13	11	4	1	3	15	5	9	0	10	14	7	6	8	2	12
S-блок 8:															
1	15	13	0	5	7	10	4	9	2	3	14	6	11	8	12

Если лучшим способом вскрытия ГОСТ является грубая сила, то это очень безопасный алгоритм. ГОСТ и спользует 256-битовый ключ, а если учитывать секретные S-блоки, то длина ключа возрастает. ГОСТ, по вид имому, более устойчив к дифференциальному и линейному криптоанализу, чем DES. Хотя случайные S-блоки ГОСТ возможно слабее фиксированных S-блоков DES, их секретность увеличивает устойчивость ГОСТ к ди ференциальному и линейному криптоанализу. К тому же, эти способы вскрытия чувствительны к количеству этапов - чем больше этапов, тем труднее вскрытие. ГОСТ использует в два раза больше этапов, чем DES, одно это возможно делает несостоятельными и дифференциальный, и линейный криптоанализ.

Другие части ГОСТ такие же, как в DES, или слабее. ГОСТ не использует существующую в DES перестановку с расширением. Удаление этой перестановки из DES ослабляет его из-за уменьшения лавинного эффекта, разумно считать, что отсутствие такой операции в ГОСТ ослабляет этот алгоритм. Сложение, используемое в ГОСТ, не менее безопасно, чем используемая в DES операция XOR.

Самым большим различием представляется использование в ГОСТ циклического сдвига вместо перестано вки. Перестановка DES увеличивает лавинный эффект. В ГОСТ изменение одного входного бита влияет на один S-блок одного этапа, который затем влияет на два S-блока следующего этапа, три блока следующего этапа, и т.д.. В ГОСТ потребуется 8 этапов прежде, чем изменение одного входного бита повлияет на каждый бит р езультата, алгоритму DES для этого нужно только 5 этапов. Это, конечно же, слабое место. Но не забывайте: ГОСТ состоит из 32 этапов, а DES только из 16.

Разработчики ГОСТ пытались достигнуть равновесия между безопасностью и эффективностью. Они изм енили идеологию DES так, чтобы создать алгоритм, который больше подходит для программной реализации. Они, по видимому, менее уверены в безопасности своего алгоритма и попытались скомпенсировать это очень большой длиной ключа, сохранением в секрете S-блоков и удвоением количества итераций. Вопрос, увенчались ли их усилия созданием более безопасного, чем DES, алгоритма, остается открытым.

14.2 CAST

CAST был разработан в Канаде Карлайслом Адамсом (Carlisle Adams) и Стаффордом Таваресом (Stafford Tavares) [10, 7]. Они утверждают, что название обусловлено ходом разработки и должно напоминать о вероя тностном характере процесса, а не об инициалах авторов. Описываемый алгоритм CAST использует 64-битовый блок и 64-битовый ключ.

CAST имеет знакомую структуру. Алгоритм использует шесть S-блоков с 8-битовым входом и 32-битовым выходом. Работа этих S-блоков сложна и зависит от реализации, подробности можно найти в литературе.

Для шифрования сначала блок открытого текста разбивается на левую и правую половины. Алгоритм сост оит из 8 этапов. На каждом этапе правая половина объединяется с частью ключа с помощью функции f, а затем XOR результата и левой половины выполняется для получения новой правой половины. Первоначальная (до этапа) правая половина становится новой левой половиной. После 8 этапов (не переставьте левую и правую п оловины после восьмого этапа) две половины объединяются, образуя шифротекст. Функция f проста:

(1) Разбейте 32-битовый вход на четыре 8-битовых части: a, b, c, d.

(2) Разбейте 16-битовый подключ на две 8-битовых половины: e, f.

(3) Подайте a на вход S-блока 1, b - на вход S-блока 2, c - на вход S-блока 3, d - на вход S-блока 4, e - на вход S-блока 5 и f - на вход S-блока 6.

(4) Выполните XOR шести выходов S-блоков, получая 32-битовый результат.

Иначе, 32-битовый вход может быть объединен с помощью XOR с 32 битами ключа, разбит на четыре 8-битовых части, которые обрабатываются S-блоками и затем объединяются с помощью XOR [7]. Безопасность N этапов, организованных таким образом, по видимому, соответствует $N + 2$ этапам другого варианта.

16-битовые подключи этапов легко получаются из 64-битового ключа. Если k_1, k_2, . . .k_8 - это 8 байтов ключа, то на этапах алгоритма используются следующие подключи:

Этап 1: k_1, k_2

Этап 2: k_3, k_4

Этап 3: k_5, k_6

Этап 4: k_7, k_8

Этап 5: k_4, k_3

Этап 6: k_2, k_1

Этап 7: k_8, k_7

Этап 8: k_6, k_5

Сила этого алгоритма заключена в его S-блоках. У CAST нет фиксированных S-блоков, для каждого прил ожения они конструируются заново. Критерии проектирования описаны в [10], изогнутыми функциями являются столбцы S-блоков, обеспечивающие необходимые свойства S-блоков (см. раздел 14.10). Созданный для данной реализации CAST S-блоков уже больше никогда не меняется. S-блоки зависят от реализации, а не от ключа.

В [10] было показано, что CAST устойчив к дифференциальному криптоанализу, а в [728] - что CAST усто йчив и к линейному криптоанализу. Неизвестно иного, чем грубая сила, способа вскрыть CAST.

Northern Telecom использует CAST в своем пакете программ Entrust для компьютеров Macintosh, PC и р абочих станций UNIX. Выбранные ими S-блоки не опубликованы. Канадское правительство считает CAST н овым стандартом шифрования. Патентная заявка на CAST находится в процессе рассмотрения.

14.3 BLOWFISH

Blowfish - это алгоритм, разработанный лично мной для реализации на больших микропроцессорах [1388, 1389]. Алгоритм незапатентован, и его код на языке C приведен в конце этой книги для широкого пользования. При проектировании Blowfish я использовал следующие критерии:

1. Скорость. Blowfish шифрует данные на 32-битовых микропроцессорах со скоростью 26 тактов на байт.
2. Компактность. Blowfish может работать менее, чем в 5 Кбайт памяти.
3. Простота. Blowfish использует только простые операции: сложение, XOR и выборка из таблицы по 32-битовому операнду. Анализ его схемы несложен, что делает при реализации алгоритма уменьшает к оличество ошибок [1391].
4. Настраиваемая безопасность. Длина ключа Blowfish переменна и может достигать 448 битов.

Blowfish оптимизирован для тех приложений, в которых нет частой смены ключей, таких как линии связи или программа автоматического шифрования файлов. При реализации на 32-битовых микропроцессорах с большим кэшем данных, таких как Pentium и PowerPC, Blowfish заметно быстрее DES. Blowfish не подходит для использования в приложениях с частой сменой ключей, например, при коммутации пакетов, или для и спользования в качестве однонаправленной хэш-функции. Большие требования к памяти делают невозможным использование этого алгоритма в интеллектуальных платах.

Описание Blowfish

Blowfish представляет собой 64-битовый блочный шифр с ключом переменной длины. Алгоритм состоит из двух частей: развертывание ключа и шифрование данных. Развертывание ключа преобразует ключ длиной до 448 битов в несколько массивов подключей, общим объемом 4168 байтов.

Шифрование данных состоит из простой функции, последовательно выполняемой 16 раз. Каждый этап с остоит из зависимой от ключа перестановки и зависимой от ключа и данных подстановки. Используются только сложения и XOR 32-битовых слов. Единственными дополнительными операциями на каждом этапе являются четыре извлечения данных из индексированного массива.

В Blowfish используется много подключей. Эти подключи должны быть рассчитаны до начала шифрования или дешифрирования данных.

P-массив состоит из 18 32-битовых подключей:

P1, P2, . . ., P18

Каждый из четырех 32-битовых S-блоков содержит 256 элементов:

$S_{1,0}$, $S_{1,1}$, . . ., $S_{1,255}$

$S_{2,0}$, $S_{2,2}$, . . ., $S_{2,255}$

$S_{3,0}$, $S_{3,3}$, . . ., $S_{3,255}$

$S_{4,0}$, $S_{4,4}$, . . ., $S_{4,255}$

Точный метод, используемый при вычислении этих подключей описан в этом разделе ниже.

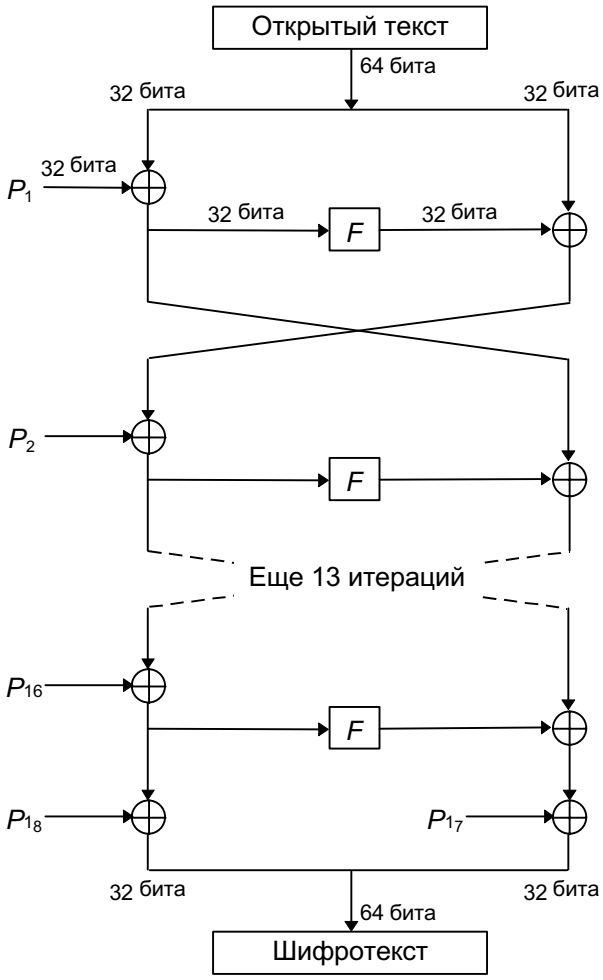


Рис. 14-2. Blowfish.

Blowfish является сетью Фейстела (Feistel) (см. раздел 14.10), состоящей из 16 этапов. На вход подается 64-битовый элемент данных x. Для шифрования:

Разбейте x на две 32-битовых половины: x_L, x_R

Для i = 1 по 16:

$x_L = x_L \oplus P_{18}$

$x_R = \mathrm{F}(x_L) \oplus x_R$

Переставить x_L и x_R (кроме последнего этапа.)

$x_R = x_R \oplus \mathrm{P}_{17}$

$x_L = x_L \oplus P_{18}$

Объединить x_L и x_R

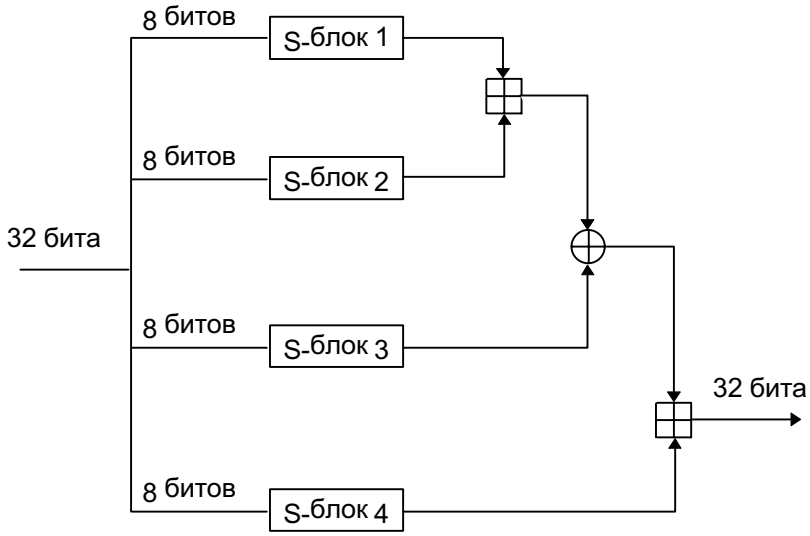


Рис. 14-3. Функция F.

Функция F представляет собой следующее (см. Рис. 14-3):

Разделить x_L на четыре 8-битовых части: a, b, c и d
$F(x_L) = ((S_{1,a} + S_{2,b} \bmod 2^{32}) \oplus S_{3,c}) + S_{4,d} \bmod 2^{32}$

Дешифрирование выполняется точно также, как и шифрование, но P_1, P_2, . . ., P_{18} используются в обратном порядке.

В реализациях Blowfish, для которых требуется очень большая скорость, цикл должен быть развернут, а все ключи должны храниться в кэше. Подробности приведены в [568].

Подключи рассчитываются с помощью специального алгоритма. Вот какова точная последовательность де йствий.

(1) Сначала P-массив, а затем четыре S-блока по порядку инициализируются фиксированной строкой. Эта строка состоит из шестнадцатиричных цифр π.

(2) Выполняется XOR P_1 с первыми 32 битами ключа, XOR P_2 со вторыми 32 битами ключа, и так далее для всех битов ключа (до P_{18}). Используется циклически, пока для всего P-массива не будет выполнена опер ация XOR с битами ключа.

(3) Используя подключи, полученные на этапах (1) и (2), алгоритмом Blowfish шифруется строка из одних нулей.

(4) P_1 и P_2 заменяются результатом этапа (3).

(5) Результат этапа (3) шифруется с помощью алгоритма Blowfish и измененных подключей.

(6) P_3 и P_4 заменяются результатом этапа (5).

(7) Далее в ходе процесса все элементы P-массива и затем по порядку все четыре S-блока заменяются вых одом постоянно меняющегося алгоритма Blowfish.

Всего для генерации всех необходимых подключей требуется 521 итерация. Приложения могут сохранять подключи - нет необходимости выполнять процесс их получения многократно.

Безопасность Blowfish

Серж Воденэ (Serge Vaudenay) исследовал Blowfish с известными S-блоками и r этапами, дифференциал ьный криптоанализ может раскрыть P-массив с помощью 2^{8r+1} выбранных открытых текстов [1568]. Для некот орых слабых ключей, которые генерируют плохие S-блоки (вероятность выбора такого ключа составляет 1 к 2^{14}), это же вскрытие раскрывает P-массив с помощью всего 2^{4r+1}. При неизвестных S-блоках это вскрытие может обнаружить использование слабого ключа, но не может определит сам ключ (ни S-блоки, ни P-массив). Это вскрытие эффективно только против вариантов с уменьшенным числом этапов и совершенно бесполезно против 16-этапного Blowfish.

Конечно, важно и раскрытие слабых ключей, даже хотя они скорее всего не будут использоваться. Слабым является ключ, для которого два элемента данного S-блока идентичны. До выполнения развертывания ключа невозможно определить, является ли он слабым. Если вы беспокоитесь об этом, вам придется выполнить ра з-

вертывание ключа и проверить, нет ли в S-одинаковых элементов. Хотя я не думаю, что это так уж необходимо.

Мне неизвестно об успешном криптоанализе Blowfish. Для безопасности не реализуйте Blowfish с умен ьшенным числом этапов.

Kent Marsh Ltd. встроила Blowfish в свой продукт обеспечения безопасности FolderBolt, предназначенный для Microsoft Windows и Macintosh. Алгоритм также входит в Nautilus и PGPfone.

14.4 SAFER

SAFER K-64 означает Secure And Fast Encryption Routine with a Key of 64 bits - Безопасная и быстрая проц едура шифрования с 64-битовым ключом [1009]. Этот не являющийся частной собственностью алгоритм, разр аботанный Джеймсом Массеем (James Massey) для Cylink Corp., используется в некоторых из продуктов этой компании. Правительство Сингапура собирается использовать этот алгоритм - с 128-битовым ключом [1010] - для широкого спектра приложений. Его использование не ограничено патентом, авторскими правами или др угими ограничениями.

Алгоритм работает с 64-битовым блоком и 64-битовым ключом. В отличие от DES он является не сетью Фейстела (см. раздел 14.10), а итеративным блочным шифром: для некоторого количества этапов применяется одна и та же функция. На каждом этапе используются два 64-битовых подключа. Алгоритм оперирует только байтами.

Описание SAFER K-64

Блок открытого текста делится на восемь байтовых подблоков: $B_1, B_2, \ldots, B_7, B_8$. Затем подблоки обрабатываются в ходе r этапов. Наконец подблоки подвергаются заключительному преобразованию. На каждом этапе используется два подключа: K_{2r-1} и K_{2r}.

На Рис. 14-4 показан один этап SAFER K-64. Сначала над подблоками выполняется либо операция XOR, л ибо сложени с байтами подключа K_{2r-1}. Затем восемь подблоков подвергаются одному из двух нелинейных пр еобразований:

$y = 45^x \bmod 257$. (Если $x = 0$, то $y = 0$.)

$y = \log_{45} x$. (Если $x = 0$, то $y = 0$.)

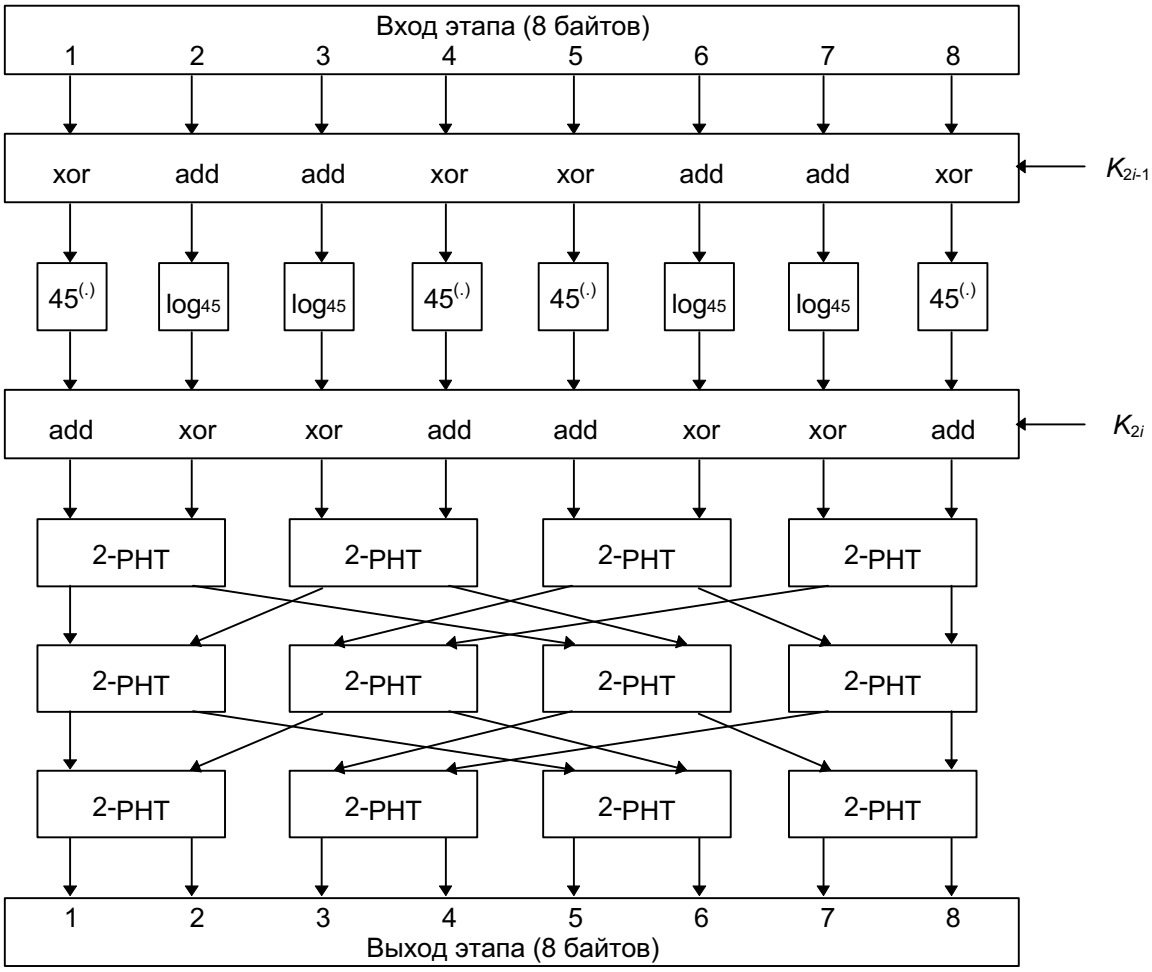


Рис. 14-4. Один этап SAFER.

Это операции в конечном поле GF(257), а 45 - элемент поля, являющийся примитивом. В реализациях SAFER K-64 быстрее выполнять поиск в таблице, чем все время рассчитывать новые результаты.

Затем подблоки либо подвергаются XOR, либо складываются с байтами подключа K_{2r}. Результат этого действия проходит через три уровня линейных операций, целью которых является увеличение лавинного эффекта. Каждая операция называется псевдоадамаровым преобразованием (Pseudo-Hadamard Transform, PHT). Если на входе PHT a_1 и a_2, то на выходе:

$b_1 = (2a_1 + a_2) \bmod 256$

$b_2 = (a_1 + a_2) \bmod 256$

После r этапов выполняется заключительное преобразование. Оно совпадает с преобразованием, являющимся первым действием каждого этапа. Над B_1, B_4, B_5 и B_8 выполняется XOR с соответствующими байтами последнего подключа, а B_2, B_3, B_6 и B_7 складываются с соответствующими байтами последнего подключа. В результате и получается шифротекст.

Дешифрирование представляет собой обратный процесс: сначала заключительное преобразование (с вычитанием вместо сложения), затем r инвертированных этапов. Обратное PHT (Inverse PHT, IPHT) - это:

$a_1 = (b_1 - b_2) \bmod 256$

$a_2 = (-b_1 + 2b_2) \bmod 256$

Массей рекомендует использовать 6 этапов, но для большей безопасности количество этапов можно увеличить.

Генерировать подключи совсем не трудно. Первый подключ, K_1, - это просто ключ пользователя. Последующие ключи генерируются в соответствии со следующей процедурой:

$K_{i+1} = (K_i <<< 3i) + c_i$

Символ "<<<" обозначает циклический сдвиг налево. Сдвиг выполняется побайтно, а c_i является константой этапа. Если c_{ij} - это j-ый байт константы i-го этапа, то можно рассчитать все константы этапов по следующей

формуле

$c_{ij} = 45^{45^{\wedge}((9i+j) \bmod 256) \bmod 257} \bmod 257$

Обычно эти значения хранятся в таблице.

SAFER K-128

Этот альтернативный способ использования ключа был разработан Министерством внутренних дел Синг апура, а затем был встроен Массеем в SAFER [1010]. В этом способе используются два ключа, K_a и K_b, по 64 бита каждый. Прием состоит в том, чтобы генерировать параллельно две последовательности подключей, а з атем чередовать подключи из каждой последовательности. Это означает, что при выборе $K_a = K_b$ 128-битовый ключ совместим с 64-битовым ключом K_a.

Безопасность SAFER K-64

Массей показал, что SAFER K-64 после 6 этапов абсолютно защищен от дифференциального криптоанализа после 8 этапов и достаточно безопасен. Уже после 3 этапов против этого алгоритма становится неэффективным и линейный криптоанализ [1010].

Кнудсен (Knudsen) обнаружил слабое место в распределении ключей: практически для каждого ключа сущ ествует не меньше одного (а иногда даже девять) другого ключа, который при шифровании какого-то другого открытого текста превращает его в тот же шифротекст [862]. Число различных открытых текстов, которые шифруются одинаковыми шифротекстами, находится в промежутке от 2^{22} до 2^{28}. Хотя такое вскрытие не может повлиять на безопасность SAFER как алгоритма шифрования, оно значительно уменьшает его надежность при использовании в качестве однонаправленной хэш-функции. В любом случае Кнудсен рекомендует использовать не меньше 8 этапов.

SAFER был спроектирован для Cylink, а Cylink были предъявлены различные обвинения со стороны NSA [80]. Я рекомендовал бы потратить несколько лет на интенсивный криптоанализ прежде, чем как-либо испол ьзовать SAFER.

14.5 3-WAY

3-Way - это блочный шифр, разработанный Джоном Дэйменом (Joan Daemen) [402, 410]. Он использует блок и ключ длиной 96 бит, и его схема предполагает очень эффективную аппаратную реализацию.

3-Way является не сетью Фейстела, а итеративным блочным шифром. У 3-Way может быть n этапов, Дэ ймен рекомендует 11.

Описание S-Way

Этот алгоритм описать несложно. Для шифрования блока открытого текста x:

For $i = 0$ to $n - 1$

$x = x$ XOR K_i

x = theta (x)

x = pi - 1 (x)

x = gamma (x)

x = pi - 2 (x)

$x = x \oplus K^{n+1}$

x = theta (x)

При этом используются следующие функции:

— theta(x) - функция линейной подстановки, в основном набор циклических сдвигов и XOR.

— pi - 1 (x) и pi - 2 (x) - простые перестановки.

— gamma (x) - функция нелинейной подстановки. Именно это действие и дало имя всему алгоритму, оно представляет собой параллельное выполнение подстановки 3-битовых данных.

Дешифрирование аналогично шифрованию за исключением того, что нужно изменить на обратный порядок битов исходных данных и результата. Исходный код, реализующий 3-Way, можно найти в конце этой книги.

Пока об успешном криптоанализе 3-Way неизвестно. Алгоритм незапатентован.

14.6 CRAB

Этот алгоритм был разработан Бертом Калиски [Burt Kaliski] и Мэттом Робшоу [Matt Robshaw] из RSA Laboratories [810]. В основе Crab лежит идея использовать методы однонаправленных хэш-функций для созд ания быстрого алгоритма шифрования. Следовательно, Crab очень похож на MD5, и в этом разделе предполаг ается, что вы знакомы с материалом раздела 18.5.

У Crab очень большой блок: 1024 байта. Так как Crab был представлен скорее как материал для исследов ания, а не реальный алгоритм, конкретной процедуры генерации ключей не было предложено. Авторы рассмо трели метод, который позволяет превратить 80-битовый ключ в три вспомогательных подключа, хотя алгоритм позволяет легко использовать ключи переменной длины. В Crab используется два набора больших подключей:

Перестановка чисел с 0 до 255: P_0, P_1, P_2, ..., P_{255}.

Массив из 2048 32-битовых чисел: S_0, S_1, S_2,..., S_{2047}.

Все эти подключи должны быть вычислены до шифрования или дешифрирования. Для шифрования 1024-байтового блока X:

(1) Разделите X на 256 32-битовых подблоков: X_0, X_1, X_2, ..., X_{255}.

(2) Переставьте подблоки X в соответствии с P.

(3) For r=0 to 3

For g = 0 to 63

$A = X_{(4g) <<< 2r}$

$B = X_{(4g+1) <<< 2r}$

$C = X_{(4g+2) <<< 2r}$

$D = X_{(4g+3) <<< 2r}$

For step s = 0 to 7

$A = A \oplus (B + f_r(B, C, D) + S_{512r+8g+s})$

$TEMP = D$

$D = C$

$C = B$

$B = A <<< 5$

$A = TEMP$

$X_{(4g) <<< 2r} = A$

$X_{(4g+1) <<< 2r} = B$

$X_{(4g+2) <<< 2r} = C$

$X_{(4g+3) <<< 2r} = D$

(4) Снова объединить X_0, X_1, X_2, ..., X_{255}, получая шифротекст.

Функции $f_r(B, C, D)$ аналогичны используемым в MD5:

$f_0(B, C, D) = (B \wedge C) \vee ((\neg B) \wedge D)$

$f_1(B, C, D) = (B \wedge D) \vee (C \wedge (\neg D))$

$f_2(B, C, D) = B \oplus C \oplus D$

$f_3(B, C, D) = C \oplus (B \vee (\neg D))$

Дешифрирование представляет собой обратный процесс. Генерирование подключей является непростой з адачей. Вот как по 80-битовому ключу K может быть сгенерирован массив перестановок P.

(1) Проинициализируйте K_0, K_1, K_2, ..., K_9 10 байтами K.

(2) For i=10 to 255

$K_i = K_{i-2} \oplus K_{i-6} \oplus K_{i-7} \oplus K_{i-10}$

(3) For i=10 to 255, $P_i = i$

(4) m=0

(5) For j=0 to 1

For i = 256 to 1 step -1

$m = (K_{256-i} + K_{257-i}) \bmod i$

$K_{257-i} = K_{257-i} <<< 3$

Переставить P_i и P_{i-1}

S-массив из 2048 32-битовых слов может быть сгенерирован аналогичным образом либо по тому же 80-битовому ключу, либо по другому ключу. Авторы предупреждают, что эти детали должны "рассматриваться только в качестве мотивации, могут быть и другие эффективные схемы, обеспечивающие лучшую безопасность" [810].

Crab был предложен как испытательный стенд для новых идей, а не как рабочий алгоритм. Во многом он использует те же приемы, что и MD5. Бихам заметил, что очень большой блок упрощает криптоанализ алгоритма [160]. С другой стороны Crab может позволять эффективно использовать очень большой ключ. В этом случае "упрощение криптоанализа" может ничего не значить.

14.7 SXAL8/MBAL

Это 64-битовый блочный алгоритм из Японии [769]. SXAL8 - это основной алгоритм, а MBAL представляет собой расширенную версию с переменной длиной блока. Так как внутри MBAL выполняется ряд умных действий, авторы утверждают, что они могут обеспечить достаточную безопасность за малое число этапов. При длине блока 1024 байта MBAL примерно в 70 раз быстрее, чем DES. К несчастью в [1174] показано, что MBAL чувствителен к дифференциальному криптоанализу, а в [865] - что он чувствителен и к линейному криптоанализу.

14.8 RC5

RC5 представляет собой блочный фильтр с большим числом параметров: размером блока, размером ключа и количеством этапов. Он был изобретен Роном Ривестом и проанализирован в RSA Laboratories [1324, 1325].

Используется три действия: XOR, сложение и циклические сдвиги. На большинстве процессоров операции циклического сдвига выполняются за постоянное время, переменные циклические сдвиги являются нелинейной функцией. Эти циклические сдвиги, зависящие и от ключа, и от данных, представляют собой интересную операцию.

RC5 использует блок переменной длины, но в приводимом примере мы остановимся на 64-битовом блоке данных. Шифрование использует $2r+2$ зависящих от ключа 32-битовых слов - S_0, S_1, S_2, ... S_{2r+1} - где r - число этапов. Эти слова мы сгенерируем позднее. Для шифрования сначала разделим блок открытого текста на два 32-битовых слова: A и B. (RC5 предполагает следующее соглашение по упаковке байтов в слова: первый байт занимает младшие биты регистра A, и т.д.) Затем:

$A = A + S_0$

$B = B + S_1$

For i = 1 to r:

$A = ((A \oplus B) <<< B) + S_{2i}$

$B = ((B \oplus A) <<< A) + S_{2i+1}$

Результат находится в регистрах A и B.

Дешифрирование также просто. Разбейте блок открытого текста на два слова, A и B, а затем:

For $i = r$ down to 1:

$B = ((B - S_{2i+1}) >>> A) \oplus A$

$A = ((A - S_{2i}) >>> B) \oplus B$

$B = B - S_1$

$A = A - S_0$

Символ ">>>" обозначает циклический сдвиг направо. Конечно же, все сложения и вычитания выполняются по модулю 2^{32}.

Создание массива ключей более сложно, но также прямолинейно. Сначала, байты ключа копируются в ма с-сив L из с 32-битовых слов, дополняя при необходимости заключительное слово нулями. Затем массив S инициализируется при помощи линейного конгруэнтного генератора по модулю 2^{32}:

$S_0 = P$

for $i = 1$ to $2(r + 1) - 1$:

$S_i = (S_{i-1} + Q) \bmod 2^{32}$

P = 0xb7e15163 и Q = 0x9e3779b9, эти константы основываются на двоичном представлении e и phi.

Наконец, подставляем L в S:

$i = j = 0$

$A = B = 0$

выполнить n раз (где n - максимум $2(r + 1)$ и c):

$A = S_i = (S_i + A + B) <<< 3$

$B = L_i = (L_i + A + B) <<< (A + B)$

$i = (i + 1) \bmod 2(r + 1)$

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

По сути, RC5 представляет собой семейство алгоритмов. Только что мы определили RC5 с 32-битовым сл о-вом и 64-битовым блоком, не существуе причин, запрещающих использовать тот же алгоритм с 64-битовым словом и 128-битовым. Для $w = 64$, P и Q равны 0xb7e151628aed2a6b и 0x9e3779b97f4a7c15, соответственно. Ривест обозначил различные реализации RC5 как RC5-w/r/b, где w - это размер слова, r - число этапов, а b - длина ключа в байтах.

RC5 является новым алгоритмом, но RSA Laboratories потрптила достаточно много времени, анализируя его работу с 64-битовым блоком. После 5 этапов статистика выглядит очень хорошо. После 8 этапов каждый бит открытого текста влияет по крайней мере на один циклический сдвиг. Дифференциальное вскрытие требует 2^{24} выбранных открытых текстов для 5 этапов, 2^{45} для 10 этапов, 2^{53} для 12 этапов и 2^{68} для 15 этапов. Конечно же, существует только 2^{64} возможных открытых текстов, поэтому такое вскрытие неприменимо против алгоритма с 15 и более этапами. Оценка для линейного криптоанализа показывает, что алгоритм безопасен после 6 этапов. Ривест рекомендует использовать не меньше 12 этапов, а лучше 16 [1325]. Это число может меняться.

RSADSI в настоящее время патентует RC5, а это названия заявлено, как торговая марка. Компания утве р-ждает, что плата за лицензирование будет очень мала, но это лучше проверить.

14.9 Другие блочные алгоритмы

Существует алгоритм, называемый в литературе CRYPTO-MECCANO [301], но он не является безопасным. Четыре японских криптографа на Eurocrypt '91 представили алгоритм, основанный на хаотичных отображениях [687, 688], Бихам выполнил криптоанализ этого алгоритма на той же конференции [157]. Другой алгоритм оп ирается на подмножества некоторого множества случайных кодов [693]. Существует множество алгоритмов, о сн-ованных на теории кодов, исправляющих ошибки: вариант алгоритма МакЭлайса (McEliece) (см. раздел 19.7) [786, 1290], алгоритм Rao-Nam [1292, 733, 1504, 1291, 1056, 1057, 1058, 1293], варианты алгоритма Rao-Nam [464, 749, 1503] и алгоритм Li-Wang [964, 1561] - все они небезопасны. CALC также небезопасен [1109]. Алг о-ритм TEA (Tiny Encryption Algorithm, Крошечный алгоритм шифрования) слишком нов, чтобы его коммент и-ровать [1592]. Другим алгоритмом является Vino [503]. MacGuffin, блочный алгоритм, предложенный Мэттом Блэйзом и мной, также небезопасен [189], он был взломан на той же конференции, на которой он был предл о-жен. BaseKing, похожий по философии на 3-way, но использующий 192-битовый блок [385], слишком нов, чт о-бы его комментировать.

Кроме того, существует множество блочных алгоритмов, разработанных вне криптографического сообщес т-ва. Некоторые из них используются различными военными и правительственными организациями. У меня нет данных о таких алгоритмах. Существуют также десятки частных коммерческих алгоритмов. Некоторые из них могут быть хороши, некоторые нет. Если компания предполагает, что опубликование ее алгоритмов не будет служить интересам компании, то лучше согласиться с ней и не использовать эти алгоритмы.

14.10 Теория проектирования блочного шифра

В раздел 11.1 я описывал принципы Шеннона для смешения и рассеяния. Спустя пятьдесят лет после того, как эти принципы были впервые сформулированы, они остаются краеугольным камнем проектирования хор о-

шего блочного шифра.

Смешение служит для маскировки взаимосвязей между открытым текстом, шифротекстом и ключом. По мните, как даже незначительная зависимость между этими тремя вещами может быть использована при дифф еренциальном и линейном криптоанализе? Хорошее смешение настолько усложняет статистику взаимосвязей, что не работают даже мощные криптографические средства.

Диффузия распространяет влияние отдельных битов открытого текста на как можно большее количество шифротекста. Это также маскирует статистические взаимосвязи и усложняет криптоанализ.

Для безопасности достаточно одного смешения. Алгоритм, состоящий из единственной зависящей от ключа таблицы соответствия 64 битов открытого текста 64 битам шифротекста был бы достаточно сильным. Проблема в том, что для такой таблицы потребовалось бы слишком много памяти: 1020 байтов. Смысл создания блочного шифра и состоит в создании чего-то похожего на такую таблицу, но предъявляющего к памяти более умеренные требования.

Прием состоит в том, чтобы в одном шифре в различных комбинациях периодически перемежать смешив ание (с гораздо меньшими таблицами) и диффузию. Это называется **результирующим шифром**. Иногда блочный шифр, который включает последовательные перестановки и подстановки, называют **сетью перестановок-подстановок** (substitution-permutation network), или **SP-сетью**.

Взгляните снова на функцию f в DES. Перестановка с расширением и P-блок реализуют диффузию, а S-блоки - смешение. Перестановка с расширением и P-блок линейны, S-блоки - нелинейны. Каждая операция сама по себе очень проста, но вместе они работают очень хорошо.

На примере DES также можно показать еще несколько принципов проектирования блочного шифра. Первым является идея **итеративного блочного шифра**. При этом предполагается, что простая функция этапа будет п оследовательно использована несколько раз. Двухэтапный DES не очень силен, чтобы все биты результата зав исели от всех битов ключа и всех битов исходных данных, нужно 5 этапов [1078, 1080]. 16-этапный DES - это сильный алгоритм, 32-этапный DES еще сильнее.

Сети Фейстела

Большинство блочных алгоритмов являются **сетями Фейстела** (Felstel networks). Эта идея датируется нач алом 70-х годов [552, 553]. Возьмите блок длиной n и разделите его на две половины длиной $n/2$: L и R. Конечно, n должно быть четным. Можно определить итеративный блочный шифр, в котором результат j-го этапа определяется результатом предыдущего этапа:

$L_i = R_{i-1}$

$R_i = L_{i-1} \oplus \mathrm{f}(R_{i-1}, K_i)$

K_i - это подключ, используемый на j-ом этапе, а f - это произвольная функция этапа.

Эту концепцию можно увидеть в DES, Lucifer, FEAL, Khufu, Khafre, LOKI, COST, CAST, Blowfish и других алгоритмах. Почему это так важно? Гарантируется, что эта функция является обращаемой. Так как для объед инения левой половины с результатом функции этапа используется XOR, следующее выражение обязательно я вляется истинным:

$L_{i-1} \oplus \mathrm{f}(R_{i-1}, K_i) \oplus \mathrm{f}(R_{i-1}, K_i) = L_{i-1}$

Гарантируется, что шифр, использующий такую конструкцию, обратим, если можно восстановить исходные данные f на каждом этапе. Сама функция f неважна, он не обязана быть обратимой. Мы можем спроектировать f настолько сложной, насколько захотим, и нам не потребуется реализовывать два различных алгоритма - один для шифрования, а другой для дешифрирования. Структура сети Фейстела автоматически позаботится об этом.

Простые соотношения

DES обладает следующим свойством: если $E_K(P) = C$, то $E_{K'}(P') = C'$, где P', C' и K' - побитовые дополнения P, C и K. Это свойство вдвое уменьшает сложность вскрытия грубой силой. Свойства комплиментарности алг оритма LOKI уменьшают сложность вскрытия грубой силой в 256 раз.

Простое соотношение можно определить как [857]:

Если $E_K(P) = C$, то $E_{\mathrm{f}(K)}(g(P,K)) = h(C,K)$

где f, g и h - простые функции. Под "простыми функциями" я подразумеваю функции, которые вычисляются легко, намного легче, чем выполнение итерации блочного шифра. В DES f представляет собой побитовое л ополнение K, g - побитовое дополнение P, а h - побитовое дополнение C. Это является результатом вкрапления ключа в часть текста с помощью XOR.

Для хорошего блочного шифра не существует простых соотношений. Методы поиска некоторых из подобных слабых мест можно найти в [917].

Групповая структура

При изучении алгоритма возникает вопрос, не образует ли он группу. Элементами группы являются блоки шифротекста для каждого возможного ключа, а групповой операцией является композиция. Изучение групповой структуры алгоритма представляет собой попытку понять, насколько увеличивается пространство шифрования при множественном шифровании.

Полезным, однако, является не вопрос о том, действительно ли алгоритм является группой, а о том, наскол ько он близок к группе. Если не хватает только одного элемента, то алгоритм не образует группу, но двойное шифрование было бы - статистически говоря - просто потерей времени. Работа над DES показала, что DES очень далек от группы. Существует также ряд интересных вопросов о полугруппе, получаемой при шифровании DES. Содержит ли она тождество, то есть, не образует ли она группу? Иными словами, не генерирует ли когда-нибудь некоторая комбинация операций шифрования (не дешифрирования) тождественную функцию? Если так, насколько длинна самая короткая такая комбинация?

Целью исследования является оценка пространства ключей для теоретического вскрытия грубой силой, а р езультат представляет собой наибольшую нижнюю границу энтропии пространства ключей.

Слабые ключи

В хорошем блочном шифре все ключи одинаково сильны. Обычно нет проблем и при алгоритме с малым количеством слабых ключей, таком как DES. Вероятность случайно выбрать один из них очень мала, такой ключ легко проверить и при необходимости отбросить. Однако, иногда эти слабые ключи могут быть задейс т-ованы, если блочный фильтр используется как однонаправленная хэш-функция (см. раздел 18.11).

Устойчивость к дифференциальному и линейному криптоанализу

Исследование дифференциального и линейного криптоанализа значительно прояснило теорию проектиров а-ния хорошего блочного шифра. Авторы IDEA ввели понятие **дифференциалов**, обобщение основной идеи х а-рактеристик [931]. Они утверждали, что можно создавать блочные шифры, устойчивые к вскрытиям такого т и-па. Результатом подобного проектирования и является IDEA [931]. Позднее это понятие было формализовано в [1181, 1182], когда Кайса Ниберг (Kaisa Nyberg) и Ларс Кнудсен (Lars Knudsen) показали, как создавать бло ч-ные шифры доказуемо безопасные по отношению к дифференциальному криптоанализу. Эта теория была ра с-ширена на дифференциалы высших порядков [702, 161, 927, 858, 860] и частичные дифференциалы [860]. К а-жется, что дифференциалы высших порядков применимы только к шифрам с малым числом этапов, но части ч-ные дифференциалы прекрасно объединяются с дифференциалами.

Линейный криптоанализ новее, и он все еще совершенствуется. Были определены понятия классификации ключей [1019] и нескольких приближений [811, 812]. Еще одно расширение криптоанализа можно найти в [1270]. В [938] была предпринята попытка объединить дифференциальный и линейный криптоанализ в одном вскрытии. Пока неясно, какая методика проектирования сможет противостоять подобным вскрытиям.

Кнудсен добился некоторого успеха, рассматривая некоторые необходимые (но, возможно, не достаточные) критерии того, что он назвал **практически безопасными сетями Фейстела** - шифров, устойчивых как к ди ф-ференциальному, так и к линейному криптоанализу [857]. Ниберг ввел для линейного криптоанализа аналог понятия дифференциалов в дифференциальном криптоанализе [1180].

Достаточно интересной кажется двойственность дифференциального и линейного криптоанализа. Эта дво й-ственность становится очевидной как при разработке методики создания хороших дифференциальных характ е-ристик и линейных приближений [164, 1018], так и при разработке критерия проектирования, обеспечивающего устойчивость алгоритмов к обоим типам вскрытия [307]. Пока точно неизвестно, куда заведет это направление исследований. Для начала Дэймен разработал стратегию проектирования алгоритма, основанную на диффере н-циальном и линейном криптоанализе [402].

Проектирование S-блоков

Сила большинства сетей Фейстела - и особенно их устойчивость к дифференциальному и линейному кри п-тоанализу - непосредственно связана с их S-блоками. Это явилось причиной потока исследований, что же обр а-зует хороший S-блок.

S-блок - это просто подстановка: отображение m-битовых входов на n-битовые выходы. Ранее я упоминал об одной большой таблице отображения 64-битовых входов на 64-битовые выходы, такая таблица представляла бы собой S-блок размером 64*64 бита. S-блок с *m*-битовым входом и n-битовым выходом называется **m*n-битовым S-блоком**. S-блоки обычно являются единственным нелинейным действием в алгоритме, именно они

обеспечивают безопасность блочного шифра. В общем случае чем S-блоки больше, тем лучше.

В DES восемь различных 6*4-битовых S-блоков. В Khufu и Khafre единственный 8*32-битовый S-блок, в LOKI 12*8-битовый S-блок, а в Blowfish и CAST 8*32-битовые S-блоки. В IDEA S-блоком по сути является умножение по модулю, это 16*16-битовый S-блок. Чем больше S-блок, тем труднее обнаружить статистические отклонения, нужные для вскрытия с использованием либо дифференциального, либо линейного криптоанализа [653, 729, 1626]. Кроме того, хотя случайные S-блоки обычно не оптимальны с точки зрения устойчивости к дифференциальному и линейному криптоанализу, сильные S-блоки легче найти среди S-блоков большего ра змера. Большинство случайных S-блоков нелинейны, невырождены и обладают сильной устойчивостью к лине йному криптоанализу - и с уменьшением числа входных битов эта доля снижается медленно [1185, 1186, 1187].

Размер m важнее размера n. Увеличение размера n снижает эффективность дифференциального криптоан ализа, но значительно повышает эффективность дифференциального криптоанализа. Действительно, если $n \geq 2^m - m$, то наверняка существует линейная зависимость для входных и выходных битов S-блока. И если $n \geq 2^m$, то линейная зависимость существует только для выходных битов [164].

Заметной частью работы по проектированию S-блоков является изучение **логических функций** [94, 1098, 1262, 1408]. Для обеспечения безопасности булевы функции, используемые в S-блоках, должны отвечать опр еделенным условиям. Они не должны быть ни линейными, ни аффинными, ни даже быть близкими к линейным или аффинным [9, 1177, 1178, 1188]. Количество нулей и единиц должно быть сбалансированным, и не должно быть никаких корреляций между различными комбинациями битов. При изменении на противоположный л юбого входного бита выходные биты должны вести себя независимо. Эти критерии проектирования также связ аны с изучением **функций изгиба**: функций, которые, как может быть показано, являются оптимально нелине йными. Хотя они определены просто и естественно, их изучение очень нелегко [1344, 1216, 947, 905, 1176, 1271, 295, 296, 297, 149, 349, 471, 298].

Очень важным свойством представляется лавинный эффект: сколько выходных битов S-блока изменяется при изменении некоторого подмножества выходных битов. Нетрудно задать для булевых функций условия, в ыполнение которых обеспечивает определенный лавинный эффект, но проектирование таких функций является более сложной задачей. **Строгий лавинный критерий** (strict avalanche criteria, SAC) обеспечивает, что с изм енением одного входного бита изменяется ровно половина выходных битов [1586]. См. также [982, 571, 1262, 399]. В одной из работ эти критерии рассматриваются в терминах утечки информации [1640].

Несколько лет назад криптографы предложили выбирать S-блоки так, чтобы таблица распределения разл ичий для каждого S-блока была однородной. Это обеспечило бы устойчивость к дифференциальному криптоан ализу за счет сглаживания дифференциалов на любом отдельном этапе [6, 443, 444, 1177]. Примером такого проектирования является LOKI. Однако такой подход иногда способствует дифференциальному криптоанализу [172]. Действительно, лучшим подходом является минимизирование максимального дифференциала. Кванджо Ким (Kwangjo Kim) выдвинул пять критериев проектирования S-блоков [834], похожих на критерии проектир ования S-блоков DES.

Выбор хороших S-блоков - не простая задача, существует множество различных идей, как лучше сделать это. Можно выделить четыре главных подхода.

1. Случайно выбрать. Ясно, что небольшие случайные S-блоки небезопасны, но большие случайные S-блоки могут оказаться достаточно хороши. Случайные S-блоки с восемью и более входами дост аточно сильны [1186, 1187]. Еще лучше 12-битовые S-блоки. Устойчивость S-блоков возрастает, если они одновременно являются и случайными, и зависящими от ключа. В IDEA используются большие зависящие от ключа S-блоки.

2. Выбрать и проверить. В некоторых шифрах свойства S-блоков, генерированных случайным образом, проверяются. Примеры такого подхода содержатся в [9, 729].

3. Разработать вручную. При этом математический аппарат используется крайне незначительно: S-блоки создаются с использованием интуитивных приемов. Барт Пренел (Bart Preneel) заявил, что "... теор етически интересные критерии недостаточны [для выбора булевых функций S-блоков] ...", и что "... н еобходимы специальные критерии проектирования" [1262].

4. Разработать математически. S-блоки создаются в соответствии с математическими законами, поэтому они обладают гарантированной надежностью по отношению к дифференциальному и линейному криптоанализу, а также хорошими диффузными свойствами. Прекрасный пример такого подхода можно найти в [1179].

Существует ряд призывов объединить "математический" и "ручной" подходы [1334], но реально, по вид имому, конкурируют случайно выбранные S-блоки и S-блоки с определенными свойствами. Конечно преимущ еством последнего подхода является оптимизация против известных методов вскрытия - дифференциального и линейного криптоанализа - но обеспечиваемая этим подходом степень защиты от неизвестных методов вскр ы-

тия также неизвестна. Разработчикам DES было известно о дифференциальном криптоанализе, и его S-блоки были оптимизированы соответствующим образом. Скорее всего, о линейном криптоанализе они не знали, и S-блоки DES очень слабы по отношению к такому способу вскрытия [1018]. Случайно выбранные S-блоки в DES были бы слабее против дифференциального криптоанализа, но сильнее против линейного криптоанализа.

С другой стороны случайные S-блоки могут не быть оптимальными по отношению к данным способам вскрытия, но они могут быть достаточно большими и, следовательно, достаточно надежными. Кроме того, они, скорее всего, будут достаточно устойчивы и против неизвестных способов вскрытия. Спор все еще кипит, но лично мне кажется, что S-блоки должны быть такими большими, насколько это возможно, случайными и зав исеть от ключа.

Проектирование блочного шифра

Проектировать блочный шифр нетрудно. Если вы рассматривает 64-битовый блочный шифр как перестано вку 64-битовых чисел, ясно, что почти все эти перестановки безопасны. Трудность состоит в проектировании блочного шифра, который не только безопасен, но также может быть легко описан и просто реализован.

Легко можно спроектировать блочный шифр, если вы используете память, достаточную для размещения S-блоков 48*32. Трудно спроектировать небезопасный вариант DES, если вы собираетесь использовать в нем 128 этапов. При длине ключа 512 битов не стоит беспокоиться о том, нет ли какой-либо зависящей от ключа ко мплиментарности.

14.11 Использование однонаправленных хэш-функций

Сымым простым способом использовать для шифрования однонаправленную хэш-функцию является хэш ирование предыдущего блока шифротекста, объединенного с ключом, а затем выполнение XOR результата с т екущим блоком открытого текста:

$C_i = P_i \oplus H(K, C_{i-1})$

$P_i = C_i \oplus H(K, P_{i-1})$

Установите длину блока равной длине результата однонаправленной хэш-функции. По сути это приводит к использованию однонаправленной хэш-функции как блочного шифра в режиме CFB. При помощи аналогичной конструкции можно использовать однонаправленную хэш-функцию и в режиме OFB:

$C_i = P_i \oplus S_i$; $S_i = H(K, C_{i-1})$

$P_i = C_i \oplus S_i = H(K, C_{i-1})$

Надежность такой схемы определяется безопасностью однонаправленной хэш-функции.

Karn

Этот метод, изобретенный Филом Карном (Phil Karn) и открытый им для свободного использования, создает обратимый алгоритм шифрования из определенных однонаправленных хэш-функций.

Алгоритм работает с 32-байтовыми блоками открытого текста и шифротекста. Длина ключа может быть произвольной, хотя определенные дины ключей более эффективны для конкретных однонаправленных хэш-функций. Для однонаправленных хэш-функций MD4 и MD5 лучше всего подходят 96-байтовые ключи.

Для шифрования сначала разбейте открытый текст на две 16-байтовых половины: P_l и P_r. Затем разбейте на две 48-байтовых половины ключ: K_l и K_r.

$P = P_l , P_r$,

$K = K_l , K_r$

Добавьте K_l к P_l и выполните хэширование однонаправленной хэш-функцией, затем выполните XOR резул ьтата с P_r, получая C_r, правую половину шифротекста. Затем, добавьте K_r к C_r выполните хэширование однон аправленной хэш-функцией. Выполните XOR результата с P_l , получая C_l. Наконец, объедините C_r и C_l , получая шифротекст.

$C_r = P_r \oplus H(P_l, K_l)$

$C_l = P_l \oplus H(C_r, K_r)$

$C = C_l, C_r$

Для дешифрирования просто инвертируйте процесс. Добавьте K_r к Cr, выполните хэширование и XOR р езультата с C_l , получая P_l. Добавьте K_l к P_l, выполните хэширование и XOR результата с C_r , получая P_r.

$P_l = C_l \oplus H(C_r, K_r)$

$P_r = C_r \oplus H(P_l, K_l)$

$P = P_l, P_r$

Общая структура Karn совпадает с структурой множества других блочных алгоритмов, рассмотренных в этом разделе. У алгоритма только два этапа, так как его сложность определяется однонаправленной хэш-функцией. А, так как ключ используется только как вход хэш-функции, он не может быть раскрыт даже при помощи вскрытия с выбранным открытым текстом, если, конечно, безопасна используемая однонаправленная хэш-функция.

Luby-Rackoff

Майкл Любы (Michael Luby) и Чарльз Ракофф (Charles Rackoff) показали, что Karn не является безопасным [992]. Рассмотрим два одноблочных сообщения: AB и AC. Если криптоаналитику известны открытый текст и шифротекст первого сообщения, а также первая половина открытого текста второго сообщения, то он может легко вычислить все второе сообщение. Хотя такое вскрытие с известным открытым текстом работает только при определенных условиях, оно представляет собой главную проблему в безопасности алгоритма.

Ее удается избежать при помощи трехэтапного алгоритма шифрования [992,1643,1644]. Он использует три различных хэш-функции: H_1, H_2 и H_3. Дальнейшие исследования показали, что H_1 может совпадать с H_2, или H_2 может совпадать с H_3, но не одновременно [1193]. Кроме того, H_1, H_2 и H_3 не могут быть основаны на итерациях одной и той же базовой функции [1643]. В любом случае при условии, что $H(k,x)$ ведет себя как псевдослучайная функция, трехэтапная версия выглядит следующим образом:

(1) Разделите ключ на две половины: K_l и K_r.

(2) Разделите блок открытого текста на две половины: L_0 и R_0.

(3) Объедините K_l и L_0 и выполните хэширование. Выполните XOR результата хэширования с R_0, получая R_1:

$R_1 = R_0 \oplus H(K_l, L_0)$

(4) Объедините K_r и R_1 и выполните хэширование. Выполните XOR результата хэширования с L_0, получая L_1

$L_1 = L_0 \oplus H(K_r, R_1)$

(5) Объедините K_l и L_1 и выполните хэширование. Выполните XOR результата хэширования с R_1, получая R_2:

$R_2 = R_1 \oplus H(K_l, L_1)$

(6) Объедините L_1 и R_2, получая сообщение.

Шифр краткого содержания сообщения

Шифр краткого содержания сообщения(Message Digest Cipher, M DC), изобретенный Питером Гутманном (Peter Cutmann) [676], представляет собой способ превратить однонаправленные хэш-функции в блочный шифр, работающий в режиме CFB. Шифр работает почти также быстро, как и хэш-функция, и по крайней мере н астолько же безопасен. Оставшаяся часть этого раздела предполаг ает знакомство с главой 18.

Хэш функции, например MD5 и SHA, используют 512-битовый текстовый блок для преобразования входн ого значения (128 битов в MD5, и 160 битов в SHA) в результат того же размера. Это преобразование необрат имо, но прекрасно подходит для режима CFB: и для шифрования, и для дешифрирования используется одна и та же операция.

Рассмотрим MDC с SHA. MDC использует 160-битовый блок и 512-битовый ключ. Используется побочный эффект хэш-функции, когда в качестве прежнего хэш-значения берется входной блок открытого текста (160 б итов), а 512-битовый вход хэш-функции играет роль ключа (см. Рис 14.5). Обычно при использовании хэш-функции для хэширования некоторого входа 512-битовый вход меняется при хэшировании каждого нового 512-битового блока. Но в данном случае 512-битовый вход становится неизменяемым ключом.

MDC можно использовать с любой однонаправленной хэш-функцией: MD4, MD5, Snefru, и т.д. Он незап атентован и может быть совершенно бесплатно использован кем угодно когда угодно и для чего угодно [676].

Однако лично я не верю в эту схему. Можно подобрать такой способ взлома, на противостояние которому хэш-функция не была рассчитана. Хэш-функции не обязаны противостоять вскрытию с выбранным открытым текстом, когда криптоаналитик выбирает некоторые начальные 160-битовые значения, получает их "зашифрованными" одним и тем же 512-битовым "ключом" и пользуется этим для получения некоторой и нформации об используемом 512-битовом ключе. Так как разработчики хэш-функций не должны беспокоиться о такой возможности, считать ваш шифр безопасным по отношению к приведенному способу вскрытия - не лу ч-

шая идея.

Безопасность шифров, основанных на однонаправленных хэш-функциях

Хотя эти конструкции и могут быть безопасными, они зависят от используемой однонаправленной хэш-функции. Хорошая однонаправленная хэш-функция не обязательно дает безопасный алгоритм шифрования. Существуют различные криптографические требования. Например, линейный криптоанализ бесполезен против однонаправленных хэш-функциях, но действенен против алгоритмов шифрования. Однонаправленная хэш-функция, такая как SHA, может обладать определенными линейными характеристиками, которые, не влияя на ее безопасность как однонаправленной хэш-функции, могут сделать небезопасным ее использование в таком алгоритме шифрования, как MDC. Мне неизвестно ни о каких результатах криптоанализа использования ко нкретной однонаправленной хэш-функции в качестве блочного шифра. Прежде чем использовать их дождитесь проведения подобного анализа.

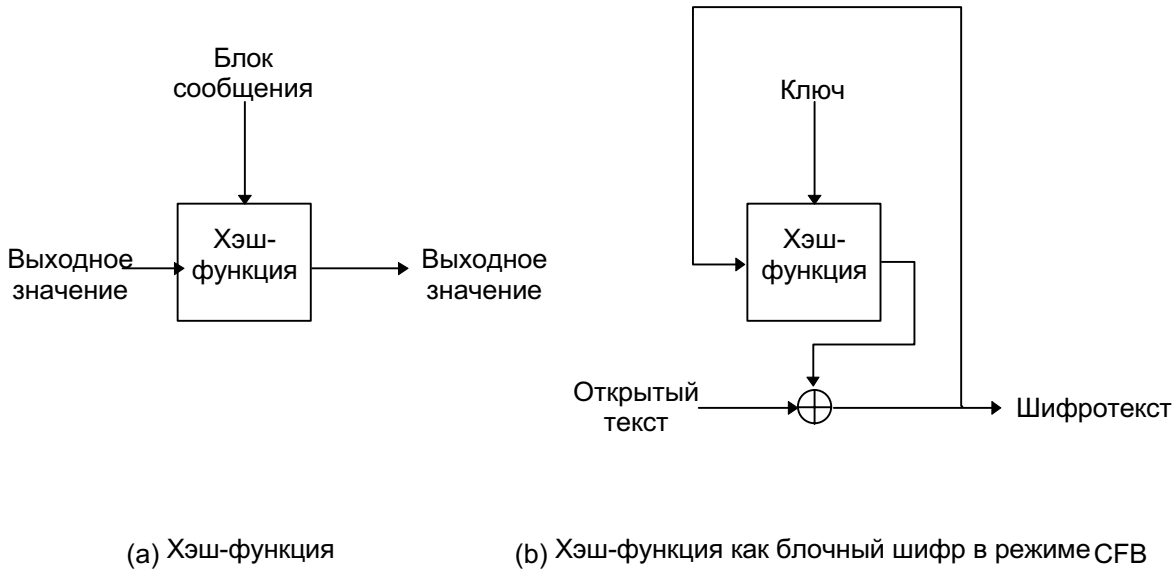


Рис. 14-5. Шифр краткого содержания сообщения (MDC).

14.12 Выбор блочного алгоритма

Это очень трудное решение. DES почти наверняка небезопасен при использовании против правительств в еликих держав, если только вы не шифруете одним ключом очень малые порции данных. Возможно этот алг оритм пока неплох против кого-нибудь другого, но вскоре и это изменится. Машины для вскрытия ключа DES грубой силой скоро станут по карману всем организациям.

Предложенные Бихамом зависимые от ключа S-блоки DES будут безопасны в течение по крайней мере н ескольких лет, может быть за исключением использования против самых хорошо обеспеченных противников. Если необходимая безопасность должна быть обеспечена на десятилетия, или вы опасаетесь криптоаналитич еских усилий правительств великих держав, воспользуйтесь тройным DES с тремя независимыми ключами.

Небеполезны и другие алгоритмы. Мне нравится Blowfish, потому что он быстр, и потому что я его прид умал. Неплохо выглядит 3-WAY, возможно все в порядке и с ГОСТом. Проблема посоветовать что-нибудь с остоит в том, что NSA почти наверняка обладает набором эффективных криптоаналитических приемов, которые до сих пор засекречены, и я не знаю, какие алгоритмы могут быть вскрыты. В Табл. 14.3 для сравнения прив едены временные соотношения для некоторых алгоритмов.

Мой любимый алгоритм - IDEA. Его 128-битовый ключ в сочетании с устойчивостью к общеизвестным средствам криптоанализа - вот источники моего теплого и нежного чувства к этому алгоритму. Этот алгоритм анализировался различными группами, и никаких серьезных замечаний не было опубликовано. В отсутствие необычайных криптоаналитических прорывов я сегодня ставлю на IDEA.

Табл. 14-3.
Скорости шифрования для некоторых блочных шифров наi486SX/33 МГц

Алгоритм	Скорость шифрования (Кбайт/с)	Алгоритм	Скорость шифрования (Кбайт/с)
Blowfish (12 этапов)	182	MDC (с MD4)	186

Blowfish (16 этапов)	135	MDC (с MD5)	135
Blowfish (20 этапов)	110	MDC (с SHA)	23
DES	35	NewDES	233
FEAL-8	300	REDOC II	1
FEAL-16	161	REDOC III	78
FEAL-32	91	RC5-32/8	127
ГОСТ	53	RC5-32/12	86
IDEA	70	RC5-32/16	65
Khufu (16 этапов)	221	RC5-32/20	52
Khufu (24 этапов)	153	SAFER (6 этапов)	81
Khufu (32 этапов)	115	SAFER (8 этапов)	61
Luby-Rackoff (с MD4)	47	SAFER (10 этапов)	49
Luby-Rackoff (с MD5)	34	SAFER (12 этапов)	41
Luby-Rackoff (с SHA)	11	3-Way	25
Lucifer	52	Тройной DES	12