Содержание

Часть I

Безопасность сообщений

Глава 4

Блочные шифры

Блочные шифры — одна из фундаментальных составляющих криптографических систем. Существует обширная литература, посвященная блочным шифрам, а сами они являются одной из наиболее хорошо изученных областей криптографии.

4.1 Что такое блочный шифр?

Блочный шифр (block cipher) — это функция шифрования, которая применяется к блокам текста фиксированной длины. Текущее поколение блочных шифров работает с блоками текста длиной 128 бит (16 байт). Такой шифр принимает на вход 128-битовый открытый текст и выдает 128-битовый шифрованный текст. Блочный шифр является обратимым: существует функция дешифрования, которая принимает на вход 128-битовый шифрованный текст и выдает исходный 128-битовый открытый текст. Открытый и шифрованный текст всегда имеет один и тот же размер, который называется *размером блока (block size)*.

Чтобы зашифровать сообщение, нужен секретный ключ. Невозможно скрыть сообщение, не сохраняя что-нибудь в секрете. Подобно открытому и шифрованному тексту, ключ шифрования также представляет собой строку битов. Наиболее распространены ключи размером 128 и 256 бит. Шифрование открытого текста p при помощи ключа K принято обозначать как $E(K, p)$ или $E_K(p)$, а расшифровку шифрованного текста c при помощи ключа K — $D(K, c)$ или $D_K(c)$.

Как правило, блочные шифры применяются для шифрования информации. К коротким сообщениям блочный шифр можно применять непосредственно. Если же длина сообщения превышает длину блока (обычно так и бы-

вает), необходимо использовать один из режимов работы блочного шифра, рассматриваемых в главе 5, “Режимы работы блочных шифров”.

Мы всегда следуем принципу Кирхгофа и предполагаем, что алгоритмы шифрования и дешифрования являются публично известными. Многим трудно смириться с таким подходом, и они предпочитают сохранять алгоритмы в секрете. Никогда не доверяйте секретным блочным шифрам (или любым другим секретным криптографическим функциям)!

Иногда блочный шифр было бы удобно представить в виде большой таблицы, построенной на основе конкретного ключа. Для каждого фиксированного ключа можно построить таблицу соответствий, которая бы отображала все варианты открытого текста в соответствующий шифрованный текст. Разумеется, это была бы *очень* большая таблица. Для блочного шифра с размером блока 32 бит понадобилась бы таблица размером 16 Гбайт, для шифра с размером блока 64 бит — 150 млн. Тбайт, а для шифра с размером блока 128 бит — $5 \cdot 10^{39}$ байт. Такому большому числу еще даже не придумали названия! Конечно же, на практике построить такую таблицу нереально, однако ее удобно использовать в качестве концептуальной модели. Мы также знаем, что блочный шифр является обратимым. Другими словами, в таблице не существует двух одинаковых элементов шифрованного текста, соответствующих разным элементам открытого текста. В противном случае функция дешифрования не смогла бы однозначно восстановить открытый текст по шифрованному тексту. Из этого можно сделать вывод, что в таблице содержится ровно по одному экземпляру всех возможных вариантов шифрованного текста. Как видите, набор элементов шифрованного текста совпадает с набором элементов открытого текста, порядок расположения которых изменен. В математике это называется *перестановкой (permutation)*. Блочный шифр с размером блока k бит задает перестановку k-битовых элементов для каждого из заданных значений ключа.

4.2 Типы атак

На первый взгляд, имея определение блочного шифра, совсем несложно дать определение безопасному блочному шифру: это блочный шифр, который позволяет сохранить открытый текст в секрете. Однако очевидно, что данное требование является необходимым, но отнюдь не достаточным для построения действительно хорошего шифра. Оно предполагает всего лишь устойчивость блочного шифра по отношению к атаке с использованием только шифрованного текста, в котором нападающий видит лишь зашифрованный текст сообщения. В литературе опубликовано несколько примеров атак такого типа [53, 91], однако они крайне редки. Большинство известных атак принадлежат

к типу атак с избранным открытым текстом. В разделе 3.6 рассматриваются основные типы атак. Все они применимы и к блочным шифрам. Существует также несколько типов атак, специфичных для блочных шифров.

Один из этих типов атак называется *атакой со связанным ключом (related-key attack)*. Впервые представленная Эли Бихемом в 1993 году [7], атака со связанным ключом предполагает, что злоумышленник имеет доступ к нескольким функциям шифрования. Все они работают с неизвестными ключами, однако эти ключи связаны определенным отношением, которое известно злоумышленнику. На первый взгляд такая атака выглядит несколько странно, однако, как оказалось, она дает весьма неплохие результаты по отношению к реальным системам. Существует множество реальных систем, которые используют разные ключи, связанные известным отношением. В одной закрытой системе для каждого нового сообщения предыдущее значение ключа увеличивалось на единицу. Таким образом, сообщения, идущие друг за другом, шифровались с помощью последовательных значений ключей. Оказывается, что подобные соотношения между ключами могут использоваться для атак на блочные шифры.

Существуют еще более экзотические типы атак. Команда разработчиков блочного шифра Twofish представила концепцию *атаки с избранным ключом (chosen key attack)*, в которой злоумышленник задает часть ключа и затем выполняет атаку со связанным ключом на оставшуюся часть ключа [85][1].

Зачем обращать внимание на какие-то неправдоподобные типы атак, такие, как атаки со связанным ключом или с избранным ключом? На это есть ряд причин. В своей практике мы видели реальные системы, которые вполне могли подвергнуться атаке со связанным ключом, поэтому данный тип атак вообще нельзя назвать неправдоподобным. Блочные шифры довольно часто используются в криптографических системах и потому подвергаются всем мыслимым и немыслимым нападениям. Одним из стандартных приемов построения функции хэширования для блочного шифра является метод Дэвиса–Мейера [95]. Оказалось, что при использовании функции хэширования Дэвиса–Мейера злоумышленник получает возможность выбирать ключ блочного шифра, что позволяет ему осуществлять атаки со связанным ключом и с избранным ключом. Любое определение безопасности блочного шифра, которое не учитывает эти или любые другие типы атак, является неполным. Блочный шифр — это модуль, который должен иметь простой интерфейс. Наиболее простым этот интерфейс будет в том случае, если он включает в себя все свойства, которые кто-либо может ожидать от блочного шифра.

[1] Дальнейшие исследования показали, что этот тип атаки не позволяет взломать Twofish [31], однако может оказаться успешным при нападении на другие блочные шифры.

Наличие некоторого несовершенства блочного шифра лишь усложняет использующую его систему многочисленными перекрестными зависимостями.

4.3 Идеальный блочный шифр

Чтобы определить безопасность блочного шифра, необходимо вначале дать определение идеальному блочному шифру. Как должен выглядеть идеальный блочный шифр? Очевидно, это должна быть случайная перестановка вариантов открытого текста. Немного уточним: для каждого значения ключа блочный шифр должен представлять собой случайную перестановку вариантов открытого текста, причем перестановки для различных вариантов ключа должны выбираться независимо друг от друга. Как уже отмечалось, 128-битовый блочный шифр (одна перестановка 128-битовых значений) можно представить себе в виде большой таблицы соответствий, содержащей 2^{128} элементов по 128 бит в каждом. В идеальном блочном шифре каждому значению ключа соответствует одна из таких таблиц, причем она выбирается случайным образом из набора всех возможных таблиц (т.е. всех возможных перестановок).

С формальной точки зрения, данное определение идеального блочного шифра является неполным, поскольку оно не задает соответствия таблиц различным значениям ключей. С другой стороны, как только будет задано соответствие таблиц, идеальный блочный шифр станет фиксированным и больше не будет случайным. Чтобы формализовать это определение, мы не можем говорить об одном конкретном идеальном шифре. Мы должны рассмотреть идеальный шифр как равномерное вероятностное распределение на множестве всех возможных блочных шифров. Имея дело с идеальным блочным шифром, необходимо размышлять в терминах вероятностей. Это приводит в восторг математиков, однако существенно усложняет и без того непростые объяснения. Поэтому будем придерживаться неформальной, но в то же время более простой концепции случайно выбранного блочного шифра.

4.4 Определение безопасности блочного шифра

В литературе дано множество определений безопасности блочного шифра (например, [52]). Большинство этих определений сформулированы с математической точки зрения, и ни одно из них не отражает аспектов, которые затронутых в предыдущих разделах. Мы же предпочитаем простое, хотя и неформальное определение.

Определение 1 *Безопасный блочный шифр — это шифр, для которого не существует атак.*

Небольшая тавтология, не так ли? Теперь необходимо определить, что такое атака на блочный шифр.

Определение 2 *Атака на блочный шифр — это нетривиальный метод обнаружения различия между блочным шифром и идеальным блочным шифром.*

Что же подразумевается под обнаружением такого различия? Речь идет о сравнении некоторого блочного шифра X с идеальным блочным шифром, имеющим такой же размер блока и такой же размер ключа. Различитель — это алгоритм, которому свойственна функция "черного ящика", вычисляющая для заданного открытого текста блочный шифр X либо идеальный блочный шифр. (Функция "черного ящика" — это функция, которую можно оценить, однако точная внутренняя структура которой неизвестна.) Алгоритму-различителю доступны и функция шифрования, и функция дешифрования. Кроме того, для каждой операции шифрования или дешифрования различитель может применять любой выбранный им ключ. Задача различителя состоит в том, чтобы определить, что именно реализует функция "черного ящика": блочный шифр X или же идеальный блочный шифр. Различителю необязательно быть совершенным — достаточно лишь, чтобы правильные ответы давались значительно чаще неправильных.

Описанная выше задача, конечно же, имеет тривиальные решения. Можно зашифровать открытый текст 0 с помощью ключа 0 и посмотреть, соответствует ли результат шифрования тому, что мы ожидаем получить от блочного шифра X. Данный алгоритм также подходит под определение различителя, однако, чтобы осуществить реальное нападение на систему, различитель должен быть нетривиальным. Именно этот аспект и затрудняет определение безопасности блочного шифра. Мы не можем формализовать понятия "тривиальный" и "нетривиальный". Это все равно что пытаться определить непристойное поведение: мы сможем понять, непристойно себя ведет человек или нет, только непосредственно столкнувшись с его поведением[2]. Различитель является тривиальным, если мы можем найти аналогичный различитель практически для каждого блочного шифра. В описанном случае различитель является тривиальным, потому что мы можем построить такой же различитель для каждого блочного шифра, даже для идеального.

Можно построить и более совершенный тривиальный различитель. Давайте зашифруем открытый текст 0 с помощью всех ключей в диапазоне $1, \ldots, 2^{32}$ и подсчитаем, как часто среди полученных результатов будет повторяться каждое конкретное значение первых 32 бит шифрованного текста.

[2] В 1964 году верховный судья США Поттер Стюарт (Potter Stewart) использовал это выражение для определения того, что следует считать непристойным поведением: "Сегодня я больше не буду пытаться определить данный тип события... но я пойму, оно это или нет, когда его увижу".

Пусть для шифра X оказалось, что значение t встречается пять раз вместо ожидаемого одного. Это свойство вряд ли характеризует идеальный шифр. Данный алгоритм также является тривиальным различителем, поскольку аналогичный метод можно применить к любому шифру X. (Маловероятно, чтобы для конкретного блочного шифра не нашлось подходящего значения t.)

Ситуация усложняется, если построить различитель следующим образом. Составим список из 1000 различных статистических показателей, которые мы можем подсчитать для шифра. Затем подсчитаем все эти показатели для шифра X и построим различитель на основе того показателя, который даст наиболее значимый результат. Мы ожидаем найти показатель с уровнем значимости, примерно равным 0,001. Разумеется, с помощью этого алгоритма можно построить различитель для любого конкретного шифра, поэтому данный метод является тривиальным, однако теперь эта тривиальность зависит не только от самого различителя, но и от того, как он был построен. Вот почему еще никому не удалось формализовать определение тривиальности и безопасности блочного шифра. Криптографическое сообщество еще не настолько хорошо знает криптографию, чтобы корректно сформулировать данное определение. Использование более формального определения, которое не учитывает целый ряд типов атак, никак не поможет в построении безопасных систем.

Допустимое количество вычислений, выполняемых различителем, должно быть ограничено. Мы не упомянули об этом в самом определении, чтобы не усложнять его. Если блочный шифр обладает явно заданным уровнем безопасности в n бит, различитель должен быть более эффективен, чем поиск путем полного перебора n-битовых значений. Если же уровень безопасности явно не указан, он принимается равным размеру ключа. Данная формулировка несколько расплывчата, однако на то есть причина. Во многих источниках просто отмечается, что различитель должен выполнить свою работу менее чем за 2^n шагов. Это, безусловно, верно, однако некоторые типы различителей дают только вероятностный результат, более похожий на поиск ключа с частичным перебором вариантов. Атака не всегда обладает прямой зависимостью между количеством проделанной работы и вероятностью обнаружить различие между шифром и идеальным шифром. Взять хотя бы такой пример: поиск путем полного перебора на половине пространства ключей требует выполнения 2^{n-1} шагов и дает правильный ответ в 75% случаев. (Если злоумышленник находит ключ, он уже знает ответ. Если же он не находит ключ, у него все еще есть 50%-ная вероятность правильно угадать этот ключ. Таким образом, общая вероятность получить правильный ответ равна $0{,}5 + 0{,}5 \cdot 0{,}5 = 0{,}75$.) Сравнивая различитель с подобным частичным поиском на пространстве ключей, мы учитываем эту особенность и не рассматриваем частичный поиск как атаку.

Наше определение безопасности блочного шифра охватывает все возможные типы атак. Атака с использованием только шифрованного текста, с известным открытым текстом, с избранным открытым текстом (в том числе и оперативная или, как ее еще называют, адаптивная), со связанным ключом и все другие типы атак реализуют нетривиальный различитель. Вот почему нам так нравится наше определение.

Вас может удивить, что мы потратили столько страниц, пытаясь дать определение безопасному блочному шифру? Это определение является очень важным, поскольку описывает простой и понятный интерфейс между блочным шифром и оставшимися частями системы. Такой тип разбивки на модули — важная особенность качественного проектирования. В системах безопасности, одним из главных врагов которых является сложность, удачная разбивка на модули имеет более существенное значение, чем в остальных областях проектирования. Если блочный шифр удовлетворяет нашему определению безопасности, мы можем считать его идеальным шифром. В конце концов, если он не ведет себя как идеальный шифр, мы можем найти для него различитель, а значит, шифр не удовлетворяет нашему требованию безопасности. Используя безопасный блочный шифр, нет необходимости помнить о его особенностях или недостатках; наш шифр будет обладать всеми свойствами, которых мы ожидаем от блочного шифра. Поскольку концепция идеального шифра очень проста, это значительно облегчает работу проектировщиков.

4.4.1 Четность перестановки

К сожалению, есть еще одно затруднение. Как уже отмечалось, шифрование блока текста при заданном значении ключа соответствует поиску соответствия в таблице перестановок. Предположим, что построение этой таблицы осуществляется в два этапа. Вначале мы инициируем таблицу, присваивая элементу с индексом i значение i. Затем создаем нужную перестановку, меняя местами два соседних элемента таблицы и повторяя эту операцию нужное количество раз. Оказывается, существует два типа перестановок: одни могут быть получены путем четного количества таких обменов (они называются четными перестановками), другие — путем нечетного количества обменов (они называются нечетными перестановками). Думаем, вы не удивитесь, узнав, что половина всех перестановок относится к четным, а половина — к нечетным.

Большинство современных блочных шифров имеют размер блока 128 бит, однако применяются к 32-битовым словам. Их функции шифрования построены на основе многократного применения 32-битовых операций. Данный метод получил заслуженное признание, однако у него есть один недостаток.

Применяя такие операции, довольно сложно получить нечетную перестановку. В результате практически все известные блочные шифры генерируют только четные перестановки.

Упомянутый выше факт позволяет построить простой различитель практически для каждого блочного шифра. Мы называем такой алгоритм *атакой с проверкой четности (parity attack)*. Для заданного значения ключа постройте перестановку, зашифровав по порядку все возможные варианты открытого текста. Если перестановка является нечетной, значит, перед нами идеальный блочный шифр, так как реальный блочный шифр никогда не генерирует нечетную перестановку. Если же перестановка четная, соответствующий блочный шифр рассматривается как реальный. Такой различитель будет давать правильные ответы в 75 случаях из ста. Он ошибется только тогда, когда ему попадается идеальный блочный шифр, генерирующий четную перестановку. Чтобы повысить вероятность получения правильного ответа, этот же алгоритм можно применить и к другим значениям ключа.

Несмотря на всю привлекательность атаки с проверкой четности, она не имеет практического применения. Чтобы определить четность перестановки, необходимо с помощью функции шифрования вычислить все пары "открытый текст–шифрованный текст", кроме одной. (Последняя пара определяется тривиально: единственный оставшийся вариант открытого текста отображается на единственный оставшийся вариант шифрованного текста.) В реальных системах никогда не следует допускать такое количество запросов к блочному шифру, поскольку успешное завершение других типов атак будет происходить еще быстрее. В частности, как только злоумышленник получит бóльшую часть пар "открытый текст–шифрованный текст", ему больше не понадобится ключ: он сможет расшифровать сообщение или хотя бы его основную часть, просто воспользовавшись таблицей соответствий для этих пар.

По определению атаку с проверкой четности можно было бы назвать тривиальной, однако это было бы не совсем честно с нашей стороны. Вместо этого мы изменим определение идеального блочного шифра и ограничим его случайно выбранными *четными* перестановками.

Определение 3 *Для каждого значения ключа идеальный блочный шифр реализует случайную четную перестановку, выбранную независимо от перестановок, соответствующих другим значениям ключа.*

Данное определение несколько усложняет нашу концепцию "идеального" шифра, однако в противном случае нам пришлось бы дисквалифицировать практически все известные науке блочные шифры. Для подавляющего большинства случаев ограничение допустимых перестановок четными перестановками является несущественным. Поскольку мы никогда не позволим злоумышленнику вычислить все пары "открытый текст–шифрованный текст",

определить различие между четными и нечетными перестановками будет невозможно.

Если у вас когда-нибудь появится блочный шифр, который *будет* генерировать нечетные перестановки, вам понадобится вернуться к первоначальному определению идеального блочного шифра. На практике атаки с проверкой четности влияют скорее на формальное определение безопасности, нежели на реальные системы, поэтому о четности перестановок можно вообще забыть.

4.5 Современные блочные шифры

На протяжении последних десятилетий мировой общественности были представлены сотни блочных шифров. Создать новый блочный шифр проще простого. Создать *хороший* новый блочный шифр чрезвычайно сложно. Мы не имеем в виду безопасность: блочный шифр по умолчанию должен быть безопасным. Хвастаться безопасностью нового шифра — это все равно что кичиться новой крышей, которая не протекает, или машиной, у которой есть фары. Самая большая сложность заключается в том, чтобы создать блочный шифр, который бы оказался эффективным во многих областях применения.

Разработка нового шифра — весьма интересное и поучительное занятие, но очень просим вас: ПОЖАЛУЙСТА, не используйте неизвестный шифр в реальных системах! Мы ни за что не станем доверять шифру до тех пор, пока его тщательно не исследуют другие эксперты. Основным требованием к новому шифру является его повсеместная публикация, однако этого недостаточно. Существует так много шифров, что лишь некоторые из них подвергаются тщательным исследованиям. Намного проще использовать один из широко известных шифров, которые уже были изучены и получили массу положительных откликов.

Практически все блочные шифры представляют собой несколько последовательных применений слабого блочного шифра, называемого *раундом (round)*. Некоторые из таких слабых раундов в совокупности образуют весьма надежный блочный шифр. Подобные структуры значительно облегчают разработку и реализацию шифра, как, впрочем, и его анализ. Большинство атак на блочные шифры начинаются с атаки на версии шифров с минимальным количеством раундов. По мере усовершенствования атаки могут применяться для нападения на шифры с все бóльшим и бóльшим количеством раундов.

Далее в главе рассматривается несколько наиболее популярных блочных шифров. Наш обзор будет не слишком обширным — полные спецификации этих шифров можно найти по указанным нами ссылкам в Internet. Мы же сконцентрируемся на общей структуре и свойствах каждого из них.

4.5.1 DES

Алгоритм шифрования DES (Data Encryption Standard — стандарт шифрования данных) [69], незаменимая рабочая лошадка криптографии, наконец-то перешагнул черту пенсионного возраста. Ограничения на размер ключа в 56 бит и размер блока в 64 бит делают DES непригодным для современных высокоскоростных компьютеров и огромных объемов данных. Он все еще может применяться в виде "тройного" DES или 3DES [72] — блочного шифра, образованного путем трех последовательных применений алгоритма DES. Это решает наиболее острую проблему малого размера ключа, однако не позволяет снять ограничение на малый размер блока. По современным стандартам DES — не особенно быстрый шифр, а 3DES работает еще в три раза медленнее. Несмотря на все это, DES до сих пор используется во многих существующих системах, однако применять DES или 3DES в новых разработках не рекомендуется.

На рис. 4.1 приведена схема одного раунда DES. Это линейная диаграмма вычислений, выполняемых в рамках алгоритма DES; подобные диаграммы часто встречаются в криптографической литературе. Каждый прямоугольник соответствует вычислению значения конкретной функции, а стрелки показывают, куда подается то или иное значение. Существует несколько стандартных соглашений по поводу обозначений на диаграммах. Операция XOR — "исключающее ИЛИ" (ее еще называют "побитовым сложением" или "сложением без переноса") — обозначается в формулах как оператор $\oplus$, а на рисунках — точно таким же символом, только большим. Иногда на схемах встречается и обычная операция сложения, которую обозначают символом $+$.

На вход алгоритма DES подается 64-битовый блок открытого текста, который разбивается на две 32-битовые половины: L (левая) и R (правая). Пе-

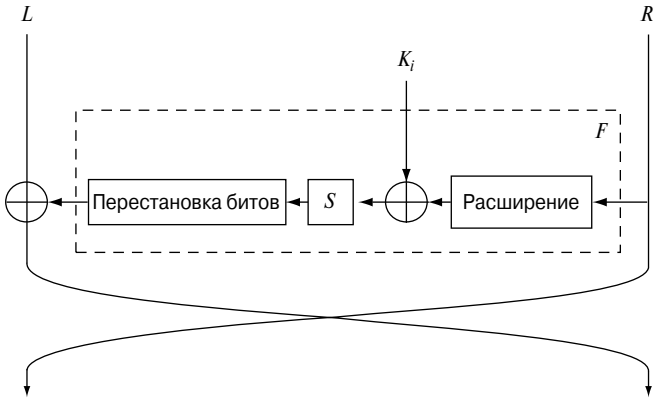

Рис. 4.1. Структура одного раунда DES

ред этим биты исходного блока текста подвергаются начальной перестановке по полуупорядоченному принципу. Никто не может толком объяснить, зачем разработчики шифра DES решили переставить биты открытого текста — ведь это не имеет никакого криптографического эффекта; однако алгоритм DES определен именно так. По окончании шифрования левая и правая половины вновь объединяются и подвергаются обратной перестановке, в результате чего вновь получается 64-битовый блок текста, однако на сей раз шифрованного.

Алгоритм DES состоит из 16 раундов, которые обозначаются цифрами от 1 до 16. Каждый раунд i преобразует пару (L, R) в новую пару (L, R) с помощью подключа K_i. Бóльшую часть преобразования выполняет функция раунда F (на рис. 4.1 она обведена штриховой рамкой). Как показано на рисунке, вначале к значению R применяется функция расширения. Она дублирует 16 битов значения R, в результате чего 32-битовое значение превращается в 48-битовое. Этот результат функции расширения складывается с помощью операции XOR с 48-битовым подключом K_i. Результат этой операции подается на вход S-матриц. По своей сути S-матрица (буква S означает *substitution*, т.е. *подстановка*) — это всего лишь таблица соответствий. Поскольку мы не можем построить таблицу соответствий для 48-битовых данных, S-матрицы состоят из восьми небольших таблиц соответствий, каждая из которых получает на вход 6 бит и выдает 4-битовый результат. Таким образом, после преобразования 48-битового значения с помощью S-матриц, мы вновь получаем 32-битовое значение. Последнее подвергается еще одной перестановке битов, после чего складывается с помощью операции XOR с левой половиной L. И наконец, значения правой и левой половины меняются местами. Эта процедура повторяется еще 15 раз.

В основе алгоритма DES лежит так называемый шифр Файстеля [29]. Идея шифра Файстеля очень проста и красива. Каждый раунд представляет собой побитовое сложение значения L со значением $F(K_i, R)$ (где F — это некоторая функция) и последующий обмен местами значений L и R. Прелесть этого алгоритма заключается в том, что расшифровка состоит из точно такого же набора операций. Необходимо поменять местами значения L и R и выполнить побитовое сложение значения L со значением $F(K_i, R)$. Это намного упрощает реализацию функций шифрования и дешифрования. Это также означает, что достаточно анализировать лишь одну из двух функций, поскольку они практически идентичны. В большинстве шифров Файстеля по окончании шифрования применяется еще один особый прием — отмена перестановки значений L и R, выполненной в последнем раунде. Благодаря этому функции шифрования и дешифрования становятся полностью идентичными за исключением порядка применения подключей. Это особенно удобно для реализации в аппаратном обеспечении, поскольку для шифрования и дешифрования может применяться одна и та же схема.

Для шифрования текста алгоритм DES использует шестнадцать 48-битовых подключей. Каждый подключ образуется путем выбора 48 бит из 56-битового ключа шифрования, причем для каждого раунда этот выбор выполняется по-своему[3].

Каждый из компонентов шифра DES имеет свое назначение. Алгоритм Файстеля упрощает структуру шифра и гарантирует перемешивание правой и левой половин текста. Сложение текста с подключом с помощью операции XOR гарантирует перемешивание ключа и данных, в чем, собственно, и заключается весь смысл шифрования. S-матрицы обеспечивают нелинейность. Без них процесс шифрования можно было бы представить в виде последовательности операций двоичного сложения, что очень легко взломать, используя методы линейной алгебры. И наконец, сочетание S-матриц, функции расширения и перестановки битов обеспечивает диффузию. Другими словами, если изменить один бит во входном значении функции F, в ее выходном значении изменится сразу несколько битов. В следующем раунде это изменение станет еще более обширным и т.п. При отсутствии диффузии незначительное изменение открытого текста приведет к незначительному изменению шифрованного текста, что можно легко отследить.

Алгоритм DES обладает рядом свойств, которые не позволяют считать его безопасным в соответствии с нашим определением безопасности. Каждый из подключей представляет собой не более чем выборку битов ключа шифрования. Если ключ шифрования равен 0, все подключи также будут равны 0. Это, в частности, означает, что все подключи будут одинаковыми. Напомним, что процедуры шифрования и дешифрования различаются лишь порядком применения подключей. Но в нашем случае все подключи будут равны. Таким образом, шифрование с помощью ключа 0 — это то же самое, что и дешифрование с помощью ключа 0. Данное свойство шифра очень легко обнаружить, а поскольку идеальный шифр таким свойством не обладает, это позволяет осуществить легкую и эффективную различающую атаку[4].

Алгоритм DES обладает и свойством комплементарности (или дополнения). Согласно этому свойству для любого ключа K и открытого текста P справедливо следующее:

$$E(\bar{K}, \bar{P}) = \overline{E(K, P)},$$

где $\bar{X}$ — это значение, каждый бит которого является дополнением соответствующего бита значения X. Другими словами, если зашифровать дополне-

[3]Выбор подключей осуществляется в соответствии с некоторым механизмом, описание которого содержится в спецификациях DES [69].

[4]Существует еще три ключа, которые обладают этим же свойством. Они называются слабыми ключами алгоритма DES.

ние открытого текста с помощью дополнения ключа, мы получим значение, которое является дополнением шифрованного (исходного) текста.

Доказать данное свойство нетрудно. Посмотрите на рис. 4.1 и подумайте, что случится, если изменить значения всех битов в L, R и K_i на противоположные. Функция расширения просто копирует биты, поэтому биты результата функции расширения также будут изменены на противоположные. У функции XOR будут изменены биты и первого и второго аргументов, поэтому их сумма останется неизменной. На вход S-матриц будет подано то же значение, что и раньше, а значит, и выходное значение не изменится. В последней операции XOR один аргумент останется неизменным, а второй будет изменен на противоположный. Поэтому в новом значении L (оно впоследствии поменяется местами с R) биты также окажутся измененными на противоположные. Другими словами, если в начале раунда заменить значения L, R и K_i их дополнениями, результатом выполнения раунда будет значение, являющееся дополнением того, которое было бы получено при использовании исходных значений. Данное свойство передается из раунда в раунд.

Идеальный блочный шифр никогда бы не обладал подобным курьезным свойством. Что еще более важно, это свойство шифра может привести к атакам на системы, использующие DES.

Иными словами, алгоритм DES больше не выдерживает проверки на профпригодность. Длина его ключа шифрования не отвечает современным требованиям. В мире уже совершено несколько успешных попыток определить ключ DES путем простого перебора вариантов.

Алгоритм 3DES работает с ключом шифрования большего размера. К сожалению, от своего предшественника DES он унаследовал и слабые ключи, и свойство комплементарности. Каждого из этих свойств вполне достаточно, чтобы по нашим стандартам шифр считался небезопасным. Кроме того, 3DES обладает ограничением на размер блока, который не может превышать 64 бит. Это существенно ограничивает объем данных, которые можно зашифровать с помощью одного ключа. (Более подробно это рассматривается в разделе 5.8.) Иногда 3DES все же приходится использовать для обеспечения совместимости с существующими системами, но будьте осторожны — он ведет себя отнюдь не как идеальный шифр.

4.5.2 AES

Несколько лет назад в США был принят новый правительственный стандарт шифрования, который получил название AES (Advanced Encryption Standard — улучшенный стандарт шифрования). Вместо того чтобы разрабатывать новый шифр или поручать эту работу конкретным людям, Национальный институт стандартов и технологий (National Institute of Standards

and Technology — NIST) обратился за помощью к криптографическому сообществу. В качестве кандидатов на лучший шифр были приняты 15 предложений [71], из которых затем отобрали пять финалистов [73]. Лучшим шифром был признан Rijndael, который и получил статус нового стандарта шифрования[5]. В целом этот процесс прошел гораздо лучше, чем ожидалось. Если кто-нибудь из вас захочет стандартизировать новую криптографическую систему, здесь определенно есть чему поучиться. Если вы получите хорошие предложения, устройте конкурс наподобие того, как это было сделано для AES, — это намного лучше, чем разрабатывать шифр всем комитетом. Если же у вас недостаточно экспертов, которые могут подавать хорошие предложения, то вам, вероятно, вообще не стоит заниматься стандартизацией.

Структура AES существенно отличается от DES. Алгоритм AES не относится к шифрам Файстеля. На рис. 4.2 показан один раунд алгоритма AES. Последующие раунды имеют аналогичную структуру. На вход алгоритма подается блок открытого текста длиной 16 байт. Вначале открытый текст складывается с помощью операции XOR с 16-байтовым (128-битовым) подключом. На рисунке этот процесс обозначен операторами $\oplus$ (каждый байт подключа складывается с соответствующим байтом открытого текста). Затем каждый из 16 байт полученного результата подается на вход таблицы S-матриц, которая отображает 8-битовые входные значения в 8-битовые выходные значения. Все S-матрицы одинаковы. Полученные байты переставляются в некотором заданном порядке. На рисунке он выглядит несколько запутанным, однако на самом деле имеет очень простую структуру. И наконец, каждая группа из 4 байт подвергается перемешиванию, которое осуществляется с помощью линейной функции перемешивания. Термин “линейная” означает лишь то, что каждый бит выходных данных функции перемешивания получен в результате применения операции XOR к нескольким входным битам.

Применение функции перемешивания завершает раунд. Полный процесс шифрования состоит из 10-14 раундов, в зависимости от размера ключа. Как и в DES, подключи AES генерируются на основе некоторого ключа шифрования, однако механизм генерации ключей полностью отличен от применявшегося в DES.

Алгоритм AES имеет свои преимущества и недостатки. Каждый шаг алгоритма состоит из нескольких операций, которые могут выполняться одновременно, что облегчает создание высокоскоростных реализаций AES. С другой стороны, операция дешифрования существенно отличается от операции шифрования. Для расшифровки текста необходимо использовать обратные S-

[5]Многие смущаются, потому что не знают, как произносить слово “Rijndael”. Не волнуйтесь: если вы не говорите по-голландски, то все равно не сможете прочитать его правильно, поэтому произносите его так, как вам больше нравится.

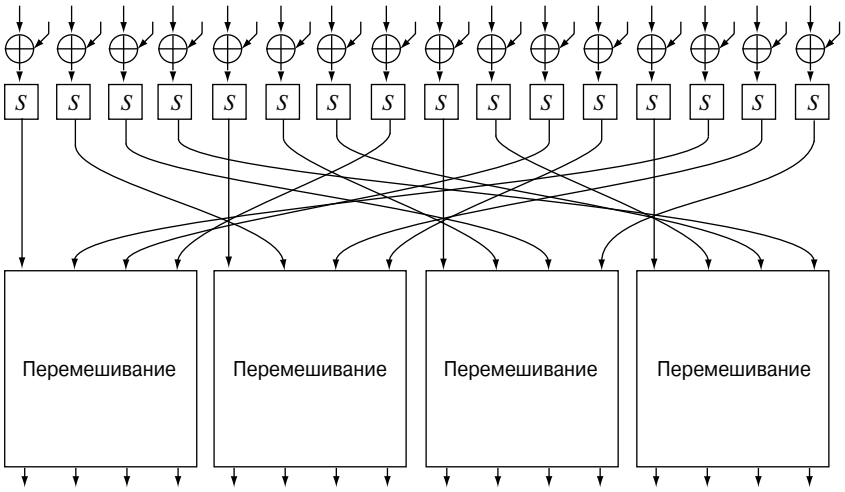

Рис. 4.2. Структура одного раунда AES

матрицы, да и функция, обратная перемешиванию, существенно отличается от самой функции перемешивания.

Как и в DES, в алгоритме AES можно выделить несколько функциональных блоков, каждый из которых имеет свое назначение. Операции XOR складывают значение ключа с данными, S-матрицы обеспечивают нелинейность, а функции перестановки и перемешивания гарантируют наличие диффузии. Шифр AES имеет очень четкую структуру, каждая часть которой выполняет строго определенную задачу.

Тем не менее полностью исключить сомнения насчет безопасности AES невозможно. Разработчики AES всегда проводили несколько агрессивную политику. На первой презентации своего алгоритма они продемонстрировали пример атаки на 6 раундов и объявили, что полный процесс шифрования должен состоять из 10–14 раундов, в зависимости от размера ключа [18]. В процессе отбора кандидатов на получение звания нового стандарта механизм атак был улучшен настолько, что стал справляться с 7 раундами алгоритма для 128-битовых ключей, 8 раундами для 192-битовых ключей и даже 9 раундами для 256-битовых ключей [30]. Казалось бы, у нас остается еще от 3 до 5 раундов на обеспечение безопасности. С другой стороны, наиболее результативная из известных атак на 128-битовые ключи покрывает уже 70% шифра. Другими словами, безопасность алгоритма AES основана на предположении, что будущие атаки на этот шифр не смогут продемонстрировать каких-либо существенных улучшений.

Предсказать будущее, как всегда, невозможно, однако иногда имеет смысл заглянуть в прошлое. До сих пор наиболее хорошо изученными шифрами яв-

лялись DES, FEAL и IDEA. В каждом из этих случаев через много лет после первой публикации шифра наблюдалось значительное усовершенствование атак на него. Время идет, и криптография тоже не стоит на месте, однако нам и сейчас остается лишь верить в то, что мы знаем об атаках все и что в ближайшее время в этой области не произойдет сколько-нибудь заметных положительных изменений.

Нельзя не отметить, что данная проблема имеет значение в основном для специалистов по криптографии. Даже если бы современные атаки были усовершенствованы настолько, что могли бы взломать AES, они, вероятно, потребовали бы около 2^{120} шагов и 2^{100} байт памяти. Этого было бы достаточно для того, чтобы по нашим стандартам считать шифр "взломанным" или, точнее, чтобы уменьшить уровень его безопасности до 120 бит. Данный шифр уже бы не удовлетворял нашим требованиям безопасности, однако терять спокойный сон из-за него мы бы не стали. Подобные атаки еще невозможно осуществить на практике и вряд ли будет возможно на протяжении тех 50 лет жизни, которые мы отвели современным криптографическим системам (см. раздел 3.7).

Гораздо большее беспокойство вызывает простая алгебраическая структура алгоритма AES [33]. Весь процесс шифрования AES можно представить в виде относительно простой замкнутой алгебраической функции с конечным полем из 256 элементов. Это еще не атака, а лишь представление, однако, если кто-нибудь когда-нибудь сможет справиться с этими функциями, AES будет взломан, что открывает абсолютно новый подход к осуществлению атак. Еще ни один из известных блочных шифров не имел такого простого алгебраического представления. Мы не знаем, приведет это к нападениям на шифр или нет, но и этого вполне достаточно, чтобы скептически относиться к использованию AES. Наше воображение не раз рисовало пренеприятнейшую картину. Проходит пять лет. Алгоритм AES применяется во многих криптографических системах по всему миру. Мы сидим в аудитории и слушаем доклад какого-то старшекурсника, имеющего отношение к совсем другой области математики. Студент откашливается и начинает говорить: "Однажды от нечего делать я начал листать книгу своего друга. Это оказалась книга по криптографии. В ней было несколько формул, которые поразительно напоминали формулы, когда-то встречавшиеся мне совсем в другом месте. Мне стало интересно, и я..." Через 20 минут этот доклад завершается словами: "Итак, мой компьютер может вычислить этот ключ примерно за два часа".

Давайте говорить откровенно. Это абсолютно несправедливая критика AES. Пока что для этого шифра не существует атак. В будущем же атаке может подвергнуться абсолютно каждый шифр, включая и AES. Тем не менее простая алгебраическая структура AES делает его потенциально уязвимым перед абсолютно новым классом атак. У криптографов еще нет опыта

в этой области. Ни один из специалистов по криптографии, к которому мы обращались, не знает, что делать с такими функциями. Возможно, в мире существуют люди, которые умеют решать задачи подобного рода, однако у нас нет никаких причин предполагать, что они знают криптографию. Давайте будем оптимистами и предположим, что вероятность хоть сколько-нибудь успешных попыток осуществления атак такого типа равна 10%. Пусть вероятность того, что эти попытки приведут к реализации практической атаки на полную версию шифра, также равна 10%. Таким образом, вероятность практической атаки на AES составляет 1% — риск, которого можно было бы избежать, используя более традиционный шифр с более сложной алгебраической структурой.

Как бы там ни было, постепенно все перейдут на использование AES, поскольку это государственный стандарт США. Мы тоже будем рекомендовать своим клиентам использовать AES, потому что он *действительно* является стандартом, а использование стандарта позволяет избежать многочисленных споров и проблем. Даже если AES когда-нибудь и взломают, никто не посмеет упрекнуть вас за то, что вы отдали предпочтение стандартному шифру. К сожалению, агрессивная политика разработчиков AES в совокупности с простой алгебраической структурой несколько ухудшает общее впечатление от нового стандарта.

4.5.3 Serpent

Шифр Serpent — еще один из пяти финалистов, соревновавшихся за право носить гордое имя стандарта AES [2]. Своей массивностью и защищенностью он напоминает танк. Наиболее консервативный из всех участников конкурса, Serpent во многом противоположен AES. В то время как разработчики алгоритма AES делали упор на красоту и эффективность, Serpent полностью ориентирован на обеспечение безопасности. Наилучшая из известных атак способна взломать только 10 из 32 раундов [6]. Недостатком шифра Serpent является его скорость — он в три раза медленнее AES. Он также не очень подходит для эффективной реализации, поскольку S-матрицы должны быть преобразованы в булевы функции, подходящие для конкретного процессора.

Кое в чем Serpent все же сходен с AES. Его алгоритм шифрования состоит из 32 раундов. В каждом раунде выполняется сложение данных и 128-битового подключа с помощью операции XOR, применение к 128-битовому значению линейной функции перемешивания и наконец параллельное применение 32 четырехбитовых S-матриц. В каждом раунде применяются 32 одинаковые S-матрицы, однако из раунда в раунд они изменяются. Кроме того, есть восемь различных S-матриц, которые поочередно используются в каждом последующем раунде.

Существует интересный прием программной реализации шифра Serpent. Обычная реализация “в лоб” работала бы слишком медленно, потому что в каждом раунде необходимо выполнять поиск соответствий в 32 S-матрицах, а таких раундов тоже 32. В сумме необходимо 1024 раза проделать поиск соответствий, а проводить операции поиска одну за другой было бы слишком медленно. Вместо этого S-матрицы представляют в виде булевых функций. Каждый из четырех выходных битов представляется как результат выполнения булевой функции от четырех входных битов. После этого процессор непосредственно вычисляет значение булевой функции, используя команды AND, OR и XOR. Хитрость заключается в том, что 32-разрядный процессор может одновременно подсчитывать значения 32 таких функций, поскольку каждая позиция двоичного разряда в регистрах процессора вычисляет значение одной и той же функции, хотя и с разными входными данными. Такой тип реализации называется *разрядно-модульным (bitslice)*. Шифр Serpent специально спроектирован в расчете на разрядно-модульную архитектуру. Помимо S-матриц, она позволяет относительно легко вычислять значения функций перемешивания.

Если бы Serpent был таким же быстрым, как Rijndael (теперешний AES), он бы практически наверняка выиграл конкурс благодаря своей консервативной структуре. Но скорость — понятие относительное. В перерасчете на зашифрованный байт Serpent оказывается почти таким же быстрым, как DES, и намного быстрее, чем 3DES. Он кажется медленным только по сравнению с другими финалистами конкурса AES.

4.5.4 Twofish

Алгоритм Twofish также вошел в число финалистов AES. Он представляет собой некий компромисс между AES и Serpent — практически такой же быстрый, как и AES, но обладающий гораздо бóльшим “запасом прочности”. Но что еще важнее, он не имеет простого алгебраического представления. Наилучшая известная нам атака способна взломать лишь 8 раундов из 16. Основным недостатком Twofish является относительная дороговизна смены ключа шифрования. Это объясняется тем, что реализация алгоритма Twofish требует выполнения целого ряда предварительных операций над ключом.

Подобно DES, алгоритм Twofish основан на шифре Файстеля. Структура Twofish представлена на рис. 4.3[6]. На вход алгоритма подается 128-битовый текст. Он разбивается на четыре 32-битовых значения, и большинство операций выполняются над 32-битовыми значениями. Как видно из рисунка,

[6]Не удивляйтесь, что этот рисунок намного больше и подробнее предыдущих. Так уж получилось, что мы принадлежим к числу разработчиков Twofish, поэтому без всяких зазрений совести взяли этот рисунок прямо из нашей книги, посвященной Twofish [85].

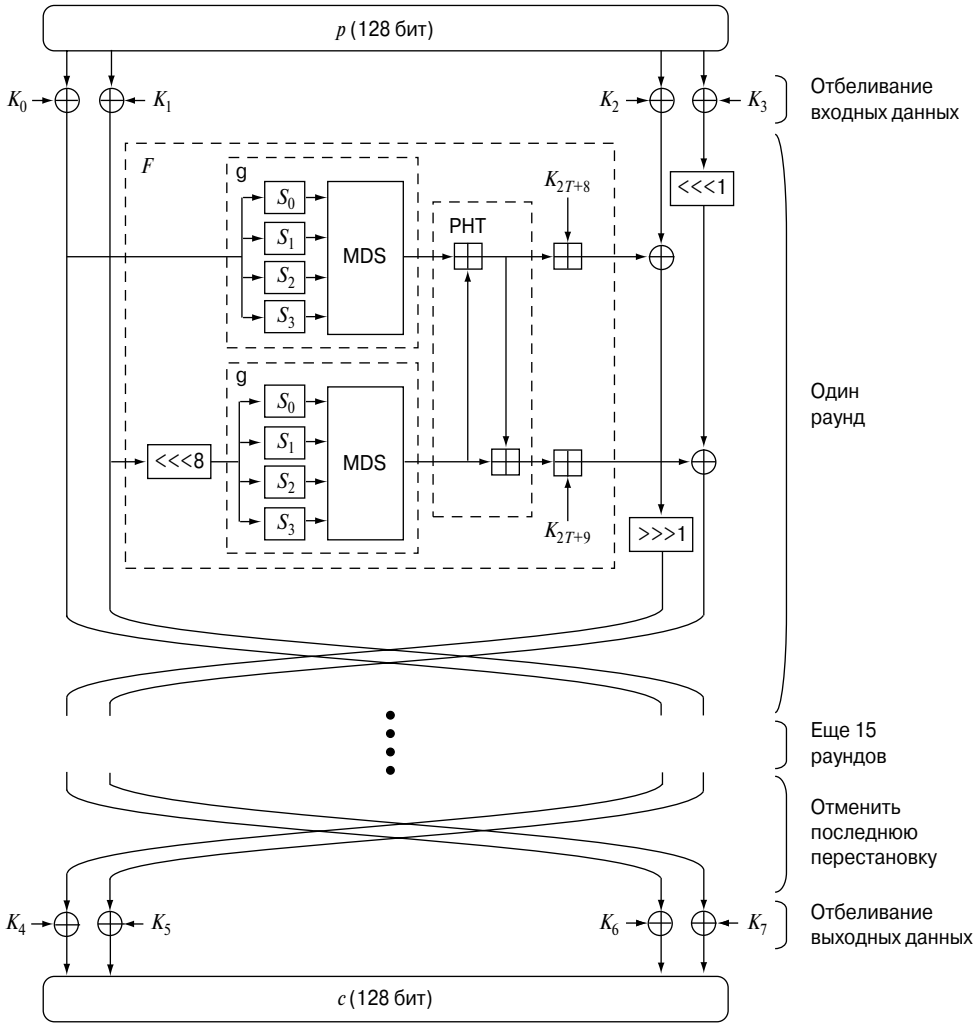

Рис. 4.3. Структура алгоритма Twofish

структура Twofish соответствует структуре шифра Файстеля. Здесь F — это функция раунда, которая состоит из двух одинаковых функций g, функции под названием PHT и операции сложения с подключом. Результат функции F складывается при помощи операции XOR с правой половиной текста (две вертикальные линии справа). Прямоугольники с символами <<< и >>> внутри означают циклический сдвиг битов 32-битового значения влево или вправо на указанное число позиций.

Каждая функция g состоит из четырех S-матриц, за которыми следует линейная функция перемешивания, очень похожая на ту, что используется

в AES. Однако S-матрицы здесь совсем другие. В отличие от всех блочных шифров, которые нам до сих пор доводилось видеть, S-матрицы Twofish не являются постоянными; их содержимое зависит от ключа. Существует алгоритм, который вычисляет S-матрицы для заданного ключа. S-матрицы были сделаны переменными, так как анализировать матрицы, зависящие от ключа, намного сложнее. Это также является причиной того, почему программные реализации Twofish выполняют ряд предварительных операций над ключом. Они вычисляют S-матрицы и сохраняют полученный результат в памяти.

Функция PHT перемешивает результаты двух функций g, используя 32-битовые операции сложения. В последней части функции F выполняется сложение данных с подключом. Обратите внимание, что обычная операция сложения обозначена как $\boxplus$, а операция "исключающее ИЛИ" — как $\oplus$.

Помимо описанных функций, в алгоритме Twofish используется так называемое *отбеливание (whitening)*. В начале и в конце шифрования данные складываются с дополнительными подключами. Это значительно затрудняет осуществление большинства типов атак, а расходы на выполнение операции отбеливания совсем небольшие.

Как и другие шифры, Twofish использует некоторый алгоритм генерации подключей раундов и двух дополнительных подключей, применяемых в начале и в конце шифрования, на основе фактического ключа шифрования.

Признаемся откровенно: наше мнение несколько субъективно. Как разработчики Twofish, мы очень любим свой шифр. Мы пытались сохранять объективность, однако в данной ситуации быть полностью объективными невозможно. Twofish был разработан частично затем, чтобы удовлетворить наши требования к хорошему блочному шифру. AES и Serpent были разработаны, чтобы удовлетворить те требования, которые казались наиболее важными их авторам. Например, в алгоритме Twofish мы сознательно добавили к шифру два однобитовых циклических сдвига. Такие сдвиги имеют два недостатка. Они вносят различие в операции шифрования и дешифрования, что требует гораздо бóльших расходов на реализацию. Они также замедляют реализацию программного обеспечения примерно на 5%. Единственная причина добавления циклических сдвигов состояла в том, чтобы нарушить ту четкую структуру разбивки на байты, которая, например, наблюдается в AES. Лично нам кажется, что алгоритм AES чрезмерно аккуратный и структурированный. Разработчики AES, в свою очередь, наверняка думают, что алгоритму Twofish не хватает структурированности и элегантности. У каждого из нас свое мнение, и мы, конечно же, глубоко уважаем мнения своих коллег. Но в этой книге все советы определяются нашим мнением.

4.5.5 Другие финалисты AES

Итак, три из пяти финалистов AES уже рассмотрены. Осталось еще два: RC6 [77] и MARS [13]. Эти алгоритмы нравятся нам гораздо меньше, а потому ограничимся лишь кратким обзором их особенностей.

Алгоритм RC6 интересен тем, что применяет умножение 32-битовых значений. В процессе состязания кандидатов на звание AES наиболее результативной атаке удалось взломать 17 раундов RC6. По сравнению с 20 раундами полной версии шифра это выглядит слишком угрожающе, чтобы ощущать себя в безопасности.

Алгоритм MARS обладает весьма запутанной структурой. В нем используется большое количество разных операций, в результате чего реализация MARS обходится дороже, чем реализация каждого из предыдущих алгоритмов. Помимо этого, нас беспокоят еще две ошибки. Ошибка в коде программы, которая генерировала S-матрицу для MARS, привела к тому, что полученная S-матрица не удовлетворяла требованиям, выдвинутыми самими разработчиками алгоритма. Но что гораздо важнее, в аргументе, приводимом авторами MARS как доказательство устойчивости алгоритма к линейному криптоанализу (особый тип атаки на шифр), обнаружилось серьезное упущение. На данный момент хорошего анализа чувствительности MARS по отношению к линейным атакам еще нет. Поскольку линейные атаки являются довольно мощным и весьма популярным инструментом нападения на шифр, такой анализ должен проводиться для каждого серьезного шифра.

Вообще-то и RC6 и MARS — неплохие шифры. Нам просто кажется, что AES, Serpent и Twofish намного лучше, поэтому советуем выбирать именно из этих трех шифров.

4.5.6 Атаки с помощью решения уравнений

Во время работы над этой книгой в мире стремительно набирал обороты новый тип атак. Он уже наделал довольно много шума в криптографическом сообществе. Основная идея этого метода заключается в том, чтобы представить блочное шифрование в виде системы линейных и квадратных уравнений над некоторым конечным полем, а затем решить эти уравнения, используя новые методы наподобие XL, FXL и XSL.

В 2002 году Николас Куртуа (Nicolas Courtois) и Йозеф Пьепжик (Josef Pieprzyk) объявили, что могут использовать указанные методы для нападения на Serpent и AES [17]. В криптографическом сообществе это заявление произвело эффект разорвавшейся бомбы. Наибольшую известность получило описание атаки на AES. Нас же гораздо больше потрясло сообщение об атаке на полную 32-раундовую версию Serpent, так как мы считали этот алгоритм самым надежным из всех кандидатов на AES.

Через некоторое время результаты Куртуа и Пьепжика получили несколько опровержений. Проблема заключается в том, что и те и другие заявления остаются чисто теоретическими. Алгоритм XSL очень сложен и слишком чувствителен к конкретной форме уравнений, для решения которых он предназначен. Оценка количества работы, необходимой для взлома AES и Serpent, построена на основе целого ряда эвристик. Так или иначе, пока что предполагаемое количество шагов намного превышает 2^{128}, поэтому наши системы, которые всегда обладают 128-битовым уровнем безопасности, могут не бояться XSL-атак. Пока XSL-атаки не претерпят значительных улучшений, они не представляют угрозы реальным системам. Тем не менее, как и все новые типы атак, они вполне могут быть усовершенствованы в самом недалеком будущем.

Насколько мы знаем, такие атаки еще не реализованы на практике. Хотелось бы увидеть компьютерную программу, которая использует эти методы для взлома сокращенных версий AES и Serpent. Получив хоть какие-то реальные данные, мы бы смогли оценить производительность алгоритма XSL по отношению к этим шифрам. Без этого мы не можем судить, действительно ли прямые атаки методом решения уравнений могут привести к взлому шифра.

Что из этого выйдет? Это не известно. Спросите нас через пять лет — возможно, тогда мы будем знать больше. На данный момент мы можем лишь высказывать свое мнение, а не приводить научно обоснованные факты. Это абсолютно новый тип атак, для которого пока что не придумано противоядия. Если XSL-атаки действительно работают, они взломают все существующие на данный момент шифры. Спасти шифр от взлома может лишь чистая случайность. С другой стороны, вполне возможно (а с нашей точки зрения, и наиболее вероятно), что XSL-атаки не применимы на практике или же применимы только к небольшому числу высокоструктурированных шифров.

А как насчет Twofish? Никто никогда не пытался применять эти методы к Twofish. Осуществить атаку подобного рода на Twofish гораздо сложнее, чем на AES или Serpent, так что никто и не пытался. Поэтому мы не знаем, насколько данные методы эффективны по отношению к нашему алгоритму. Если кому-нибудь удастся применить XSL-атаку для взлома AES или Serpent, она, без сомнения, будет применена и к Twofish.

4.5.7 Какой блочный шифр выбрать

Вот в чем вопрос. Не забывайте, что наше мнение несколько субъективно, потому что мы принадлежим к команде разработчиков Twofish. Мы также провели массу времени, пытаясь взломать другие шифры из числа финалистов AES, что еще сильнее повлияло на нашу точку зрения.

Заявления о возможности осуществления XSL-атак делают процесс выбора блочного шифра несколько неприятным. Мы просто не знаем, насколько такие атаки влияют на безопасность блочного шифра. Информации по этой теме пока еще очень немного, а та, что есть, постоянно оспаривается. Поскольку ни один блочный шифр не разрабатывался с учетом возможности XSL-атак, все эти шифры могут оказаться уязвимыми. На наш взгляд, в текущей ситуации выбор шифра не должен определяться восприимчивостью к XSL-атакам. Никто не может сказать, насколько один шифр более уязвим, нежели другой.

Самым безопасным выбором для карьеры, безусловно, является AES. Это официальный стандарт шифрования, принятый правительством США. Он будет использоваться всеми и каждым. Конечно же, мы не думаем, что это самый удачный выбор в плане безопасности данных. Тем не менее, даже если AES и взломают, это будет не ваша вина. Вы, вероятно, не раз слышали фразу: “Никого не оштрафуют за то, что он купил IBM”. Точно так же никто не упрекнет вас за то, что вы выбрали AES. Итак, если речь не идет о вашем семейном бюджете и/или спокойном сне, смело выбирайте AES.

Алгоритм AES имеет и другие преимущества. Он довольно прост в реализации и применении. Его поддерживают все криптографические библиотеки и любят все клиенты, потому что “это стандарт”. Исходя из этого, вы определенно не ошибетесь, выбрав AES.

Если вы серьезно обеспокоены безопасностью своих данных и готовы ради этого пожертвовать быстродействием, выбирайте Serpent. В процессе отбора кандидатов на AES все авторитетные криптографы признали, что Serpent был самым безопасным (или самым консервативным) из всех поступивших предложений.

Приведенные выше аргументы практически не оставляют места для Twofish. Выбирайте Twofish только тогда, когда хотите получить скорость AES без всех присущих тому недостатков в плане обеспечения безопасности. Разумеется, в этом случае все преимущества AES как официального стандарта шифрования будут направлены против вас. Если Twofish взломают, вас обвинят в неправильном выборе шифра.

В некоторых ситуациях наилучшим выбором все еще является 3DES. Если вам нужно обеспечить обратную совместимость с существующими системами или же если остальные части системы не поддерживают размеры блоков более 64 бит, выбирайте 3DES. Не забывайте, однако, что он не удовлетворяет нашим критериям безопасности, и будьте особенно бдительны, используя маленькие 64-битовые блоки.

4.5.8 Каким должен быть размер ключа

Все три шифра, рекомендуемые нами для использования в современных системах (AES, Serpent и Twofish), поддерживают ключи размером 128, 192 и 256 бит. Практически всем приложениям достаточно уровня безопасности 128 бит. Несмотря на это, нам очень не нравятся 128-битовые ключи.

Эти ключи хороши всем, за исключением одного: возможность осуществления атак на основе коллизий. Нам то и дело попадаются системы, чувствительные к атакам, в основе которых лежит парадокс задачи о днях рождения, или к двусторонним атакам. Одни разработчики просто игнорируют такие атаки, другие думают, что защищены от них, однако ни те, ни другие не застрахованы от появления новых, более изощренных способов нападения. Большинство режимов работы блочных шифров в той или иной форме допускают двусторонние атаки. Мы уже достаточно натерпелись от них, а потому предлагаем еще одно правило проектирования.

Правило проектирования 3. *Если уровень безопасности системы равен n бит, каждое криптографическое значение должно иметь длину, как минимум, $2n$ бит.*

Это правило сводит на нет все усилия по обнаружению коллизий, а расходы на его реализацию не так уж велики. В реальной жизни, однако, следовать ему довольно сложно. Например, для обеспечения 128-битового уровня безопасности необходимо использовать блочный шифр с размером блока 256 бит, однако все распространенные блочные шифры работают с блоками по 128 бит. Данная проблема намного серьезнее, чем кажется. Существует довольно большое количество атак на основе коллизий, направленных на режимы работы блочных шифров.

Несмотря на это, мы, как минимум, можем использовать ключи большего размера, которые поддерживают все рекомендуемые нами шифры. Отсюда урок: используйте 256-битовые ключи!

К сожалению, шифр AES (Rijndael) — единственный из всех финалистов AES, который работает с 256-битовыми ключами медленнее, чем со 128-битовыми[7]. Как следствие этого, во избежание потери производительности вас могут вынудить использовать ключи меньшего размера. Мы вовсе не утверждаем, что 128-битовые ключи небезопасны как таковые. Вполне возможно создать систему со 128-битовым уровнем безопасности, используя 128-битовые ключи, однако это будет крайне сложно. Например, вы не сможете просто воспользоваться одним из стандартных режимов работы блочных шифров.

[7] Serpent работает с одной и той же скоростью независимо от размера ключа. Twofish работает медленнее при выполнении предварительных вычислений над ключами большего размера, однако в программной реализации скорость шифрования не зависит от размера ключа.

Вам придется вводить дополнительные операции сложения данных с подключами, чтобы помешать осуществлению атак на основе коллизий. Это и есть тот вид сложности, который ведет к появлению "слабых мест". Поэтому рекомендуем использовать именно 256-битовые ключи.

Обратите внимание, что мы рекомендуем использовать 256-битовые ключи в системах с уровнем безопасности 128 бит. Другими словами, эти системы спроектированы так, чтобы выдержать атаку, состоящую из 2^{128} операций. Просто запомните, что для выбора размера параметров оставшейся части системы нужно использовать значение уровня безопасности (128 бит), а не размера ключа (256 бит).

Нельзя не отметить еще одно затруднение. Мы используем 256-битовый ключ для разработки систем со 128-битовым уровнем безопасности. Уже упоминавшиеся нами XSL-атаки применяются к алгоритмам AES и Serpent с ключами большего размера. Как утверждается, эти атаки более эффективны, чем поиск путем полного перебора *ключей большего размера*. Данное утверждение касается атаки на Serpent, которая требует выполнения более чем 2^{128} шагов. Даже если XSL-атака, как это и было заявлено, сработает против 256-битового шифра Serpent, она не будет атакой на 256-битовый Serpent, если установить уровень безопасности Serpent (как и всей остальной системы) равным 128 бит. В конце концов, если уровень безопасности равен 128 бит, тогда атака на шифр должна быть более эффективной, чем поиск путем полного перебора 2^{128} элементов, а текущие XSL-атаки на Serpent требуют выполнения гораздо большего количества шагов. Это же справедливо и для XSL-атак на AES.