Часть 2

Криптографические методы

Глава 7
Длина ключа

7.1 Длина симметричного ключа

Безопасность симметричной криптосистемы является функцией двух факторов: надежности алгоритма и длины ключа. Первый более важен, но роль второго легче продемонстрировать.

Пусть надежность алгоритма совершенна. На практике этого чрезвычайно трудно достигнуть, но в примере - достаточно легко. По совершенством я подразумеваю отсутствие лучшего пути взлома криптосистемы, чем вскрытие грубой силой с помощью перебора всех возможных ключей.

Для выполнения такого вскрытия криптоаналитику требуется кусочек шифротекста и соответствующего о т-крытого текста, вскрытие грубой силой представляет собой вскрытие с известным открытым текстом. Для блочного шифра криптоаналитику понадобится блок шифротекста и соответствующий открытый текст: обычно 64 бита. Заполучить такие кусочки открытого текста и шифротекста легче, чем можно себе представить. Криптоаналитик может получить каким-то образом копию открытого текста сообщения и перехватить соответствующий шифротекст. Он может знать что-то о формате шифротекста: например, что это файл в формате WordPerfect, или у него есть стандартный заголовок сообщения электронной почты, или файл каталога UNIX, или изображение в формате TIFF, или стандартная запись в базе данных клиентов. Все эти форматы содержат некоторые предопределенные байты. Криптоаналитику для такого вскрытия не нужно много открытого текста.

Рассчитать сложность вскрытия грубой силой нетрудно. Если используется 8-битовый ключ, то существует 2^8, или 256, возможных ключей. Следовательно, для обнаружения правильного ключа потребуется, самое бол ьшее, 256 попыток, с 50-процентной вероятностью найти нужный ключ после половины попыток. Если длина ключа равна 56 битам, то существует 2^{56} возможных ключей. Если компьютер может проверить миллион кл ючей в секунду, поиск нужного ключа займет в среднем 2285 лет. Если используется 64-битовый ключ, то тому же суперкомпьютеру понадобится около 585000 лет, чтобы найти правильный ключ среди 2^{64} возможных ключей. Если длина ключа равна 128 битам поиск ключа займет 10^{25} лет. Возраст вселенной составляет всего 10^{10} лет, поэтому 10^{25} лет - это большое время. При 2048-битовом ключе миллион компьютеров, работая параллел ьно и проверяя миллион ключей в секунду, потратят 10^{587} лет в поисках ключа. К этому времени вселенная давно расширится, превратившись в ничто или сожмется.

Прежде чем кидаться изобретать криптосистему с 8-килобайтным ключом, вспомните, что другой стороной является надежность: алгоритм должен быть настолько безопасен, чтобы лучшего способа, чем вскрывать его грубой силой, не существовало. Это не так просто, как может показаться. Криптография - это тонкое искусство. Выглядящие совершенными криптосистемы часто оказываются чрезвычайно слабыми. Пара изменений, внесенных в сильные криптосистемы, может резко ослабить их. Криптографам-любителям следует подвергать по чти параноидальному сомнению каждый новый алгоритм. Лучше доверять алгоритмам, над которыми годами бились профессиональные криптографы, не сумев взломать их, и не обольщаться утверждениями конструкторов алгоритмов об их грандиозной безопасности.

Вспомните важный момент из раздела 1.1: безопасность криптосистем должна основываться на ключе, а не особенностях алгоритма. Предположим, что криптоаналитику известны все подробности вашего алгоритма. Предположим, что у него есть столько шифротекста, сколько ему нужно, и что он попытается выполнить инте нсивное вскрытие с использованием только шифротекста. Предположим, что он попытается выполнить вскрытие с использованием открытого текста, имея в своем распоряжении столько данных, сколько ему нужно. Предположим даже, что он попытается выполнить вскрытие с использованием выбранного открытого текста. Если ваша криптосистема останется безопасной даже перед лицом всех подобных опасностей, то... у вас действительно что-то есть.

Несмотря на это предупреждение пространство, предоставляемое криптографией для маневра, достаточно велико. В действительности, безопасность такого типа во многих практических ситуациях не нужна. У большинства врагов нет таких знаний и вычислительных средств, как у больших правительств, а тем, кто обладает такими возможностями, может оказаться ненужным взламывать вашу криптосистему. Если вы организуете заговор с целью свергнуть большое правительство, проверенные и правильные алгоритмы, приведенные в конце этой книги, будут для вас жизненно необходимы. А все остальные пусть просто получат удовольс твие.

Оценки времени и стоимости вскрытия грубой силой

Вспомните, что вскрытие грубой силой обычно является вскрытием с использованием известного открытого текста, для этого нужно немного шифротекста и соответствующего открытого текста. Если вы предполагаете, что наиболее эффективным способа взлома алгоритма является вскрытие грубой силой - большое допущение - то ключ должен быть достаточно длинным, чтобы сделать вскрытие невозможным. Насколько длинным?

Скорость вскрытия грубой силой определяется двумя параметрами: количеством проверяемых ключей и скоростью проверки одного ключа. Большинство симметричных алгоритмов в качестве ключа могут использовать в качестве ключа любую битовую последовательность фиксированной длины. Длина ключа DES составляет 56 бит, всего может быть 2^{56} возможных ключей. Длина ключей для ряда алгоритмов, обсуждаемых в этой книге, равны 64 битам, всего может быть 2^{64} возможных ключей. Другие алгоритмы используют 128-битовые ключи.

Скорость, с которой может быть проверен каждый ключ, имеет менее важное значение. Для проводимого анализа я предполагаю, что скорость проверки ключа для каждого алгоритма примерно одинакова. В действительности скорость проверки одного алгоритма может быть в два, три или даже десять раз выше чем другого. Но так как для тех длин ключей, для которых мы проводим поиск, время поиска в миллионы раз больше, чем время проверки одного ключа, небольшие отличия в скорости проверки не имеют значения.

В криптологической среде большинство споров по поводу вскрытия грубой силой сконцентрированы вокруг алгоритма DES. В 1977 году Уитфилд Диффи и Мартин Хеллман [497] сформулировали условия существования специализированной машины по взлому DES. Эта машина состоит из миллионов микросхем, каждая из которых проверяет миллион ключей в секунду. Такая машина за два часа сможет проверить 2^{56} за 20 часов. При вскрытии алгоритма с 64-битовым ключом проверка всех 2^{64} потребует 214 дней.

Задача вскрытия грубой силой как будто специально придумана для параллельных процессоров. Каждый процессор проверяет подмножество пространства ключей. Процессорам не нужно обмениваться между собой информацией, единственным используемым сообщением будет сообщение, сигнализирующее об успехе. Не требуется и доступ к одному участку памяти. Сконструировать машину с миллионом процессоров, каждый из которых работает независимо от других, нетрудно.

Сконструировать машину для взлома грубой силой Майкл Винер решил [1597, 1598]. (Он сконструировал машину для DES, но анализ может быть выполнен почти для всех алгоритмов.) Он разработал специализированные микросхемы, платы и стойки, оценил затраты и сделал вывод, что за миллион долларов можно построить машину, которая сможет взломать 56-битный ключ DES key в среднем за 3.5 часа (и наверняка за 7 часов). Соотношение стоимость/скорость является линейным. Для ряда длин ключей эти значения обобщены в 6-й. Вспомните о законе Мура: мощь вычислительных средств приблизительно каждые 18 месяцев. Это означает, что затраты будут уменьшаться на порядок каждые пять лет, и то, что в 1995 году стоит миллион долларов, в 2000 году будет стоить около 100000 долларов. Еще более упростить процесс вычислений могла бы конвейеризация [724].

Для 56-битовых ключей эти суммы оказываются вполне по карману большинству крупных корпораций и многим криминальным организациям. Военные бюджеты большинства промышленно развитых стран могут позволить взламывать и 64-битные ключи. Вскрытие 80-битного ключа все еще за пределами возможного, но если текущая тенденция сохранится, то через каких-нибудь тридцать лет все может измениться.

Конечно, нелепо прогнозировать компьютерную мощь на 35 лет вперед. Технологические прорывы, популярные в научной фантастике, могут сделать эти прогнозы смешными. С другой стороны, неизвестные в настоящее время физические ограничения могут сделать эти прогнозы нереально оптимистичными. В криптографии умнее быть пессимистом. Применение в алгоритме 80-битного ключа кажется недостаточно дальновидным. Используйте ключ, длина которого, по меньшей мере, 112 бит.

Табл. 7-1.
Оценки среднего времени для аппаратного вскрытия грубой силой в 1995 году.

	Длина ключей в битах					
Стоимость	40	56	64	80	112	128
$100 К	2 секунды	35 часов	1 год	70000 лет	10^{14} лет	10^{19} лет
$1 М	0.2 секунды	3.5 часа	37 дней	7000 лет	10^{13} лет	10^{18} лет
$10 М	0.02 секунды	21 минута	4 дня	700 лет	10^{12} лет	10^{17} лет
$100 М	2 миллисекунды	2 минуты	9 часов	70 лет	10^{11} лет	10^{16} лет
$1 Г	0.2 миллисекунды	13	1 час	7 лет	10^{10} лет	10^{15} лет
$10 Г	0.02. миллисекунды	1 секунда	5.4 минуты	245 дней	10^{9} лет	10^{14} лет
$100 Г	2 микросекунды	0.1 секунды	32 секунд	24 дня	10^{8} лет	10^{13} лет
$1 Т	0.2 микросекунды	0.01 секунды	3 секунды	2.4 дня	10^{7} лет	10^{12} лет
$10 Т	0.02 микросекунды	1 миллисекунда	0.3 секунды	6 часов	10^{6} лет	10^{11} лет

Если взломщик очень сильно хочет взломать ключ, все, что ему нужно, это потратить деньги. Следовательно, стоит попытаться определить минимальную "цену" ключа: в пределах какой стоимости сведений можно пользоваться одним ключом прежде, чем его вскрытие станет экономически выгодным ? Крайний случай: если шифрованное сообщение стоит $1.39, то нет финансового смысла устанавливать аппаратуру стоимостью 10 миллионов долларов для взлома этого ключа. С другой стороны, если стоимость открытого текста - 100 миллионов долларов, то дешифрирование этого одиночного сообщения вполне окупит стоимость аппаратуры взлома. Кроме того, стоимость многих сообщений со временем очень быстро падает.

Программное вскрытие

Без специализированной аппаратуры и огромных параллельных машин вскрытие грубой силой намного сложнее. Программное вскрытие в тысячи раз медленнее, чем аппаратное.

Реальная угроза программного вскрытия грубой силой страшна не своей неизбежностью, а тем, что такое вскрытие "свободно". Ничего не стоит загрузить простаивающий микрокомпьютер проверкой возможных ключей. Если правильный ключ будет найден - замечательно, если нет - ничего не потеряно. Ничего не стоит использовать для этого целую сеть микрокомпьютеров. В недавних экспериментах с DES 40 рабочих станций в течение одного дня сумели проверить 2^{34} ключей [603]. При этой скорости для проверки всех ключей потребуется четыре миллиона дней, но если попытки вскрытия будут предприняты достаточным количеством людей, то кому-нибудь где-нибудь повезет. Как было сказано в [603]:

> Основной угрозой программного вскрытия является слепое везение. Представьте себе университетскую сеть из 512 объединенных в сеть рабочих станций. Для некоторых университетских городков это сеть весьма среднего размера. Такие сети могут даже расползтись по всему миру, координируя свою деятельность по электронной почте. Пусть каждая рабочая станция способна работать (с алгоритмом) со скоростью 15000 шифрований в секунду. ... С учетом накладных расходов на проверку и смену ключей уменьшим скорость до . . . 8192 проверок в секунду на машину. Чтобы, используя описанную систему, исчерпать пространство (56-битовых) ключей потребуется 545 лет (в предположении, что сеть тратит на эту задачу 24 часа в сутки). Заметим, однако, что с помощью таких вычислений сторонники нашего студента получают один шанс из 200000 раскрыть ключ в течение одного дня. За долгий уикенд их шансы возрастают до одного из шестидесяти шести тысяч. Чем быстрее их аппаратура, или чем больше задействовано машин, тем лучше становятся их шансы. Вероятность заработать на жизнь, выигрывая на скачках, невысока, но разве не эти выигрыши заполняют собой пресс-релизы. К примеру, это гораздо большая вероятность, чем возможность выигрыша в правительственных лотереях. "Один на миллион"? "Один раз за тысячу лет "? Больше невозможно с полной ответственностью делать такие заявления. Является ли приемлемым этот продолжающийся риск?

Использование алгоритма с 64-битовым ключом вместо 56-битового ключа делает это вскрытие в 256 раз сложнее. А 40-битовый ключ делает картину просто безрадостной. Сеть из 400 компьютеров с производительностью 32000 шифрований в секунду может за день выполнить вскрытие грубым взломом 40-битового ключа. (В 1992 году алгоритмы RC2 и RC4 было разрешено экспортировать с 40- битовым ключом - см. раздел 13.8.)

128-битовый ключ делает нелепой даже мысль о вскрытии грубым взломом. По оценке промышленных экспертов к 1996 году в мире будет использоваться 200 миллионов компьютеров. Эта оценка включает все - лт гигантского мэйнфрейма Cray до блокнотных компьютеров. Даже если все эти компьютеры будут брошены на вскрытие грубой силой, и каждый из них будет выполнять миллион шифрований в секунду, время раскрытия ключа все равно будет в миллион раз больше времени существования вселенной.

Нейронные сети

Нейронные сети не слишком пригодны для криптоанализа, в первую очередь из-за формы пространства решений. Лучше всего нейронные сети работают с проблемами, имеющими непрерывное множество решений, одни из которых лучше других. Это позволяет нейронным сетям обучаться, предлагая все лучшее и лучшие решения. Отсутствие непрерывности в алгоритме почти не оставляет места обучению : вы либо раскроете ключ, либо нет. (По крайней мере, это верно при использовании любого хорошего алгоритма .) Нейронные сети хорошо работают в структурированных средах, где обучение возможно, но не в высокоэнтропийном, кажущемся случайным мире криптографии.

Вирусы

Самая большая трудность в получении миллионов компьютеров для вскрытия грубым взломом - это убедить миллионы компьютерных владельцев принять участие во вскрытии. Вы могли бы вежливо попросить, но это требует много времени, и они могут сказать нет. Вы могли бы пробовать силой ворваться в их компьютеры, но это потребует еще больше времени и может закончиться вашим арестом. Вы могли бы также использовать компьютерный вирус, чтобы распространить программу взлома среди как можно большего количества компьютеров.

Эта особенно коварная идея впервые появилась в [1593]. Взломщик пишет и выпускает на волю компьютерный вирус. Этот вирус не переформатирует жесткий диск, не удаляет файлы, но во время простоя компьютера он работает на криптоаналитической проблемой грубого взлома. Различные исследования показали, что комп ь-

ютер простаивает от 70 до 90 процентов времени, так что у вируса не будет проблем с временем для решения этой задачи. Если он нетребователен и в других отношениях, то его работа даже не будет заметна.

В конце концов, одна из машина наткнется на правильный ключ. В этот момент имеются два варианта пр одолжения. Во первых, вирус мог бы породить другой вирус. Он не делал бы ничего, кроме самовоспроизведения и удаления всех найденных копий вскрывающего вируса, но содержал бы информацию о правильном ключе. Этот новый вирус просто распространялся бы среди компьютеров, пока не добрался бы до компьютера челов ека, который написал первоначальный вирус.

Другим, трусливым подходом бал бы вывод на экран следующего сообщения :

> В этом компьютере есть серьезная ошибка. Пожалуйста позвоните 1-8001234567 и продиктуйте оператору следующее 64-битовое число:
>
> xxxx xxxx xxxx xxxx
>
> Первому, кто сообщит об этой ошибке будет выплачено вознаграждение 100 долларов.

Насколько эффективно такое вскрытие? Пусть типичный зараженный компьютер проверяет тысячу ключей в секунду. Эта скорость намного меньше потенциальных возможностей компьютера, ведь мы полагаем, что он иногда будет делать и другие вещи. Предположим также, что типичный вирус инфицирует 10 миллионов машин. Этот вирус может вскрыть 56-битовый ключ за 83 дня, а 64 битовый - за 58 лет. Вам возможно пришлось бы подкупить разработчиков антивирусного программного обеспечения, но это уже ваши проблемы . Любое увеличение скорости компьютеров или распространения вируса, конечно, сделало бы это нападение более эффективным.

Китайская лотерея

Китайская Лотерея - эклектический, но возможный способ создания громадной параллельной машины для криптоанализа [1278]. Вообразите, что микросхема, вскрывающая алгоритм грубой силой со скоростью милл ион проверок в секунду, встроена в каждый проданный радиоприемник и телевизор. Каждая микросхема запр ограммирована для автоматической проверки различного набора ключей после получения пары открытый текст/шифротекст по эфиру. Каждый раз когда китайское правительство хочет раскрыть ключ, оно передает исходные данные по радио. Все радиоприемники и телевизоры в стране начинают пыхтеть. В конечном счете, правильный ключ появляется на чьем-нибудь дисплее. Китайское правительство платит приз тому человеку - это гарантирует, что результат будет сообщен быстро и правильно, и также способствует рыночному успеху р адиоприемников и телевизоров с микросхемами вскрытия.

Если у каждого человека в Китае, будь то мужчина, женщина или ребенок, есть радиоприемник или телев изор, то правильное значение 56-битового ключа появится через 61 секунду. Если радиоприемник или телевизор есть только у каждого десятого китайца(что близко к действительности), то правильный ключ появится через 10 минут. Правильный 64-битовый ключ будет раскрыт через 4.3 часа (43 часа, если радиоприемник или телевизор есть только у каждого десятого китайца).

Чтобы сделать такое вскрытие возможным на практике, необходимо сделать ряд модификаций. Во первых, проще, чтобы каждая микросхема проверяла случайные, а не уникальные ключи . Это сделает вскрытие на 39% медленнее, что не очень важно для чисел такого масштаба. Затем, Китайская коммунистическая партия должна принять решение, что каждый должен включать свой приемник или телевизор в определенное время, чтобы гарантировать работу всех приемных устройств во время передачи пары открытый текст/шифротекст . Наконец, каждому должно быть приказано позвонить в Центр - или как он там называется - когда ключ появляется у него на экране и зачитать строку чисел, появившуюся на экране .

Эффективность Китайской лотереи для различных стран и различных длин ключа показана в 5-й. Ясно, что Китай оказался бы в лучшем положении, если бы у каждого китайца - мужчины, женщины или ребенка - бал свой приемник или телевизор. В Соединенных штатах живет меньше людей, но гораздо больше аппаратуры . Штат Вайоминг самостоятельно сможет взломать 56-битовый ключ меньше, чем за день.

Табл. 7-2.
Оценки среднего времени вскрытия грубой силой при китайской лотерее

(Все данные взяты из *World Almanac and Book of Facts* за 1995 год.)

Страна	Население	Количество телевизоров/радиоприемников	Время взлома	
			56 бит	64 бита
Китай	1190431000	257000000	280 секунд	20 часов
США	260714000	739000000	97 секунд	6.9 часа
Ирак	19890000	4730000	4.2 часа	44 дня
Израиль	5051000	3640000	5.5 часа	58 дней
Вайоминг	470000	1330000	15 часов	160 дней
Виннемукка, Невада	6100	17300	48 дней	34 года

Биотехнология

Если возможно создание биомикросхем, то было бы глупо не попытаться использовать их в инструмента криптоанализа вскрытием грубой. Рассмотрим гипотетическое животное, называемое "DESозавром" [1278]. Оно состоит из биологических клеток, умеющих проверять возможные ключи. Пары "открытый текст/шифротекст" поступают в клетки по некоторому оптическому каналу (видите ли, все эти клетки прозрачны). Решения доставляются к органам речи DESозавра с помощью специальных клеток, путешествующих по кровеносной системе животного.

Типичный динозавр состоит из 10^{14} клеток (без бактерий). Если каждая из них выполняет миллион шифрований в секунду (неплохой результат), вскрытие 56-битового ключа займет семь десятитысячных секунды. Вскрытие 64-битового ключа потребует меньше, чем две десятых секунды. Вскрытие 8-битового ключа все же продлится 10^{11} лет.

Другой биологический подход состоит в использовании генетически проектируемых криптоаналитических морских водорослей, которые умеют выполнять вскрытие криптографических алгоритмов грубой силой [1278]. Такие организмы, покрыв большую область, позволили бы создать распределенную машину с большим количеством процессоров. Пара "открытый текст/шифротекст" могла бы передаваться по радио через спутник. Обнаружение результата организмом могло бы стимулировать близлежащие ячейки изменить цвет, сообщая решение обратно на спутник.

Предположим, что типичная клетка морской водоросли - это кубик со стороной 10 микрон (возможно, это оценка сверху, следовательно 10^{15} клеток заполнят кубический метр. Выплесните их в океан, покрывая 200 квадратных миль (518 квадратных километров) на глубину один метр (это ваши проблемы, как осуществить это - я подаю только идею), и у вас будет 10^{23} водорослей (более чем сотней миллиарда галлонов), плавающих в океане. (Для сравнения, из танкера *Valdez* вытекло 10 миллионов галлонов нефти.) Если каждая из них может проверять миллион ключей в секунду, то для 128-битового алгоритма они раскроют ключ в только спустя 100 лет. (Возникшее при этом цветение морских водорослей - это ваша проблема.) Крупные достижения в быстродействии морских водорослей, их диаметр или даже размеры пятна в океане могут заметно уменьшить эти значения. Даже не спрашивайте меня о нанотехнологии.

Термодинамические ограничения

Одним из следствий закона второго термодинамики является то, что для представления информации необходимо некоторое количество энергии. Запись одиночного бита, изменяющая состояние системы, требует количества энергии не меньше чем kT; где T - абсолютная температура системы и k - постоянная Больцмана. (Не волнуйтесь, урок физики уже почти закончен.)

Приняв, что $k = 1.38*10^{-16}$ эрг/K, и что температура окружающей вселенной 3.2K, идеальный компьютер, работая при 3.2K, потреблял бы $4.4*10^{-16}$ эрга всякий раз, когда он устанавливает или сбрасывает бит. Работа компьютера при температуре более низкой, чем температура космического пространства, потребовала бы дополнительных расходов энергии для отвода тепла.

Далее, энергия, излучаемая нашим Солнцем за год, составляет около $1.21*10^{41}$ эргов. Это достаточно для выполнения $2*10^{56}$ перемен бита в нашем идеальном компьютере, а этого, в свою очередь, хватит для того, чтобы 187-битовый счетчик пробежал все свои значения. Если мы построим вокруг Солнца сферу Дайсона и перехватим без всяких потерь всю его энергию за 32 года, мы сможем получить компьютер для вычисления 2^{192} чисел. Конечно, энергии для проведения каких-нибудь полезных вычислений с этим счетчиком уже не

останется.

Но это только одна жалкая звезда. При взрыве типичной сверхновой выделяется около 10^{51} эргов. (В сто раз больше энергии выделяется в виде нейтрино, но пусть они пока летают.) Если всю эту энергию удастся бросить на одну вычислительную оргию, то все свои значения сможет принять 219-битовый счетчик.

Эти числа не имеют ничего общего с самой аппаратурой, они просто показывают максимальные значения, обусловленные термодинамикой. Кроме того, эти числа наглядно демонстрируют, что вскрытие грубой силой 256-битового ключа будет невозможно, пока компьютеры построены из обычной материи и располагаются в обычном пространстве.

7.2 Длина открытого ключа

Однонаправленные функции обсуждались в разделе 2.3. Однонаправленной функцией является умножение двух больших простых чисел, получить произведение, перемножив числа, нетрудно, но нелегко разложить произведение на множители и получить два больших простых числа (см. раздел 11.3). Криптография с открытыми ключами использует эту идею для создания однонаправленной функции с люком. На самом деле, это ложь, *не доказано*, что разложение на множители является тяжелой проблемой (см. раздел 11.4). Насколько сегодня известно, это похоже на правду. И даже если это так, никто не может доказать, что трудные проблемы действительно трудны. Все считают, что разложение на множители является трудной задачей, но это никогда не было доказано математически.

На этом стоит остановиться поподробнее. Легко представить, что лет через 50 мы соберемся вместе, вспоминая старое доброе время, когда все люди считали, что разложение на множители было трудным и лежало в основе криптографии, а различные компании делали из этого деньги. Легко вообразить, что будущие достижения в теории чисел упростят разложение на множители или достижения теории сложности сделают разложение на множители тривиальным. Нет причин верить в это - и большинство людей, знающих достаточно, чтобы иметь собственное мнение, скажет вам, что подобное развитие событий является маловероятным - но нет и причин верить, что такого прорыва не случится.

В любом случае, доминирующие сегодня алгоритмы шифрования с открытым ключом основаны на трудности разложения на множители больших чисел, которые являются произведением двух больших простых чисел. (Другие алгоритмы основаны на так называемой Дискретной проблемой логарифма, но пока предположим, что к ней применимы те же рассуждения.) Эти алгоритмы также восприимчивы к вскрытию грубой силой, но по разному. Взлом этих алгоритмов состоит не из перебора всех возможных ключей, а из попыток разложения больших чисел на множители (или взятия дискретных логарифмов в очень большом конечной области - точно такая же проблема). Если число слишком мало, вы никак не защищены. Если число достаточно велико, то вы надежно защищены против всей вычислительной мощи мира, если она будет биться над этой задачей с настоящего времени до тех пор, пока Солнце не станет сверхновой - таково сегодняшнее понимание математики этой проблемы. В разделе 11.3 разложение на множители рассматривается математически подробно, а здесь я ограничусь оценкой времени разложения на множители чисел различной длины.

Разлагать большие числа на множители нелегко, но, к несчастью для проектировщиков алгоритмов, этот процесс становится все легче. Что еще хуже, он становится легче ч большей скоростью, чем предсказывалось математиками. В 1976 году Ричард Гай (Richard Guy) писал: "Я был бы немало удивлен, если бы кто-нибудь научился разлагать на множители произвольные числа порядка 10^{80} в течение данного столетия" [680]. В 1977 году Рон Ривест (Ron Rivest) заявил, что разложение на множители 125-разрядного числа потребует 40 квадриллионов лет [599]. В 1994 году было разложено на множители 129-разрядное число [66]. Если из этого и можно сделать какие-то выводы, то только то, что предсказывать глупо.

В 4-й приведены результаты разложения на множители за последнюю дюжину лет. Самым быстрым алгоритмом разложения на множители является квадратичное решето (см. раздел 11.3).

Эти числа сильно пугают. Сегодня 512-битовые числа уже используются в операционных системах. Разложение их на множители, и полная компрометация, таким образом, системы защиты, вполне реально. Червь в Internet мог бы сделать это в течение уикенда.

Табл. 7-3.
Разложение на множителя с помощью "квадратичного решета"

Год	Число десятичных разрядов в разложенном числе	Во сколько раз сложнее разложить на множители 512-битовое число
1983	71	>20 миллионов
1985	80	>2 миллионов
1988	90	250000
1989	100	30000
1993	120	500
1994	129	100

Вычислительная мощь обычно измеряется в mips-годах: годовая работа компьютера, выполняющего миллион операций в секунду (one-million-instruction-per-second, mips), или около $3*10^{13}$ операций. Принято, что машина с производительностью 1 mips-год эквивалентна VAX 11/780 компании DEC. То есть, mips-год - это год работы компьютера VAX 11/780 или эквивалентного. (100 МГц Pentium - это машина в 50 mips, а Intel Paragon из 1800 узлов - примерно 50000 mips.)

В 1983 году разложение на множители 71-разрядного числа требовало 0.1 mips-года, в 1994 году разложение на множители 129-разрядного числа потребовало 5000 mips-лет. Такой взлет вычислительной мощности обусловлен, в основном, введением распределенных вычислений, использующих время простоя сети рабочих станций. Этот подход был предложен Бобом Силверманом (Bob Silverman) и полностью разработан Аржаном Ленстрой (Arjen Lenstra) и Марком Манассом (Mark Manasse). В 1983 году разложение на множители использовало 9.5 часов процессорного времени на единственном компьютере Cray X-MP, в 1994 году разложение на множители заняло 5000 mips-лет и использовало время простоя 1600 компьютеров во всем мире в течение приблизительно восьми месяцев. Современные методы разложения на множители позволяют использовать подобные распределенные вычисления.

Картина даже продолжает ухудшаться. Новый алгоритм разложения на множители - решето общего поля чисел - заменил квадратичное решето. В 1989 году математики сказали бы вам, что решето общего поля чисел никогда не будет иметь практического значения. В 1992 году они сообщили бы, что оно реализуемо, но дает выигрыш по сравнению с квадратичным решетом только для чисел со 130-150 десятичными разрядами и больших. Сегодня известно, что этот новый алгоритм быстрее, чем квадратичное решето, для чисел значительно меньших, чем 116-разрядные [472, 635]. Решето общего поля чисел может разложить на множители 512-битовое число в 10 раз быстрее, чем квадратичное решето. На 1800-узловом компьютере Intel Paragon выполнение этого алгоритма заняло бы меньше года. В 3rd показано количество mips-лет, которое требуется для разложения чисел различных размеров при использовании современных реализаций решета общего поля чисел [1190].

Табл. 7-4.
Разложение на множители с помощью решета общего поля чисел

Количество бит	Сколько mips-лет нужно для разложения
512	30000
768	$2*10^{8}$
1024	$3*10^{11}$
1280	$1*10^{14}$
1536	$3*10^{16}$
2048	$3*10^{20}$

Кроме того, решето общего поля чисел становится все быстрее и быстрее. Математики изобретают новые трюки, оптимизации, методы, и нет причин считать, что эта тенденция оборвется. Близкий алгоритм, решето специального поля чисел, уже может разлагать на множители числа определенной специализированной формы - обычно не используемые в криптографии - гораздо быстрее, чем решето общего поля чисел может разложить на

множители любые числа того же размера. Разумно предположить, что решето общего поля чисел может быть оптимизировано, чтобы достичь такой же скорости [1190], возможно, что NSA уже знает, как это сделать. В 2-й показано количество mips-лет, требуемое для разложения чисел различной длины при помощи решета специального поля чисел [1190].

Табл. 7-5.
Разложение на множители с помощью решета специального поля чисел

Количество бит	Сколько mips-лет нужно для разложения
512	<200
768	100000
1024	$3*10^7$
1280	$3*10^9$
1536	$2*10^{11}$
2048	$4*10^{14}$

В 1991 году участники семинара Европейского института безопасности систем (European Institute for System Security) согласились, что 1024-битовых модулей будет достаточно для длительного хранения секретов до 2002 года [150]. Однако, они предупреждали: "Хотя участники этого семинара являются лучшими специалистами в соответствующих областях, это заявление (по поводу срока безопасности) должно быть воспринято с осторожностью." Это хороший совет.

Умный криптограф сверхконсервативен при выборе длин открытых ключей. Чтобы понять, насколько длинный ключ вам нужен, вам придется оценить нужную безопасность и время жизни ключа, не забывая о текущем состоянии искусства разлагать на множители. Чтобы получить тот же уровень безопасности, который давало 512-битовое число в начале восьмидесятых, сегодня вам понадобится 1024-битовое число. Если же вы хотите, чтобы ваши ключи оставались безопасными в ближайшие 20 лет, 1024-битовое число, по видимому, слишком коротко.

Даже если ваши конкретные секреты не стоят усилий, нужных для разложения вашего модуля, вы можете оказаться в опасности. Представьте себе автоматическую банковскую систему, использующую для безопасности RSA. Мэллори может предстать перед судом и заявить: "Читали ли вы в газете за 1994 год, что RSA-129 был взломан, и что 512-битовые числа могут быть разложены на множители любой организацией, которая может потратить несколько миллионов долларов и подождать несколько месяцев ? Мой банк использует для безопасности 512-битовые числа и, между прочим, эти семь изъятий сделаны не мной." Даже если Мэллори лжет, судья, вероятно, может потребовать, чтобы банк это доказал.

Почему не использовать 10000-битовые ключи? Конечно, можно, но чем длиннее ваши ключи, тем больше стоимость вычислений. Вам нужен ключ, который был бы достаточно длинным для обеспечения безопасности, но достаточно коротким, чтобы его можно было использовать.

Ранее в этом разделе я называл предсказания глупостью. Теперь я сам попытаюсь предсказать кое-что. В 1-й приведены мои рекомендации по выбору длин открытых ключей в зависимости от того, какой срок безопасности ключа вам нужен. Для каждого года приведены три длины ключа, одна для частного лица, одна для крупной корпорации и одна для правительства большого государства.

Вот некоторые соображения из [66]:

> Мы считаем, что сможем получить доступ к 100 тысячам компьютеров без сверхчеловеческих усилий и неэтичных действий. То есть, мы *не собираемся* выпускать в Internet "червя" или разрабатывать вирус, который бы предоставил бы нам вычислительные ресурсы. Во многих организациях многие тысячи машин подключены к сети. Доступ к их возможностям потребует искусной дипломатии, но не является невозможным. Приняв среднюю производительность машины равной 5 mips и время работы 1 год, вполне возможно осуществить проект, который требует полмиллиона mips-лет.

Проект разложения на множители 129-разрядного числа без значительных усилий смог задействовать 0.03 процента оценочной полной вычислительной мощности Internet [1190]. Разумно предположить, что хорошо разрекламированный проект получит на год 2 процента всемирной вычислительной мощности.

Предположим, что увлеченный криптоаналитик сможет получить в свое распоряжение 10000 mips-лет, большая корпорация - 10^7 mips-лет, а правительство большой страны - 10^9 mips-лет. Предположим также, что вычислительная мощь будет возрастать на порядок каждые пять лет. И , наконец, предположим также, что ус-

пехи в математике разложения на множители позволят нам раскладывать любые числа со скоростью, сравнимой с той, которую обеспечивает решето специального поля чисел. (Это пока невозможно, но прорыв может случиться в любой момент.) 1st рекомендует для различных лет использовать с целью обеспечения безопасности различные длины ключей.

Табл. 7-6.
Рекомендованные длины открытых ключей в (битах)

Год	Частное лицо	Корпорация	Правительство
1995	768	1280	1536
2000	1024	1280	1536
2005	1280	1536	2048
2010	1280	1536	2048
2015	1536	2048	2048

Не забывайте учитывать значимость ключа. Открытые ключи часто используются для длительной обеспечения безопасности важной информации: главный ключ банка для системы электронных наличных, ключ, используемый правительством для подтверждения паспортов, ключ цифровой подписи государственного нотариуса. Возможно, не стоит тратить месяцы машинного времени на вскрытие какого-то личного ключа, но если можете с помощью добытого ключа напечатать собственные деньги, то идея становится весьма захватывающей. Длина 1024-битовогой ключа достаточна для подписи чего-нибудь, что будет проверено в течение недели, месяца, даже нескольких лет. Но вы же не хотите, представ перед судом лет 20 спустя с подписанным электронным образом документом, смотреть, как противоположная сторона показывает, как подделать документы, используя эту же подпись.

Предсказывать более далекое будущее еще глупее. Кто может знать, каких успехов к 2020 году достигнет вычислительная техника, сетевые вычисления и математика? Однако, если окинуть взглядом всю картину, можно заметить, что в каждом следующем десятилетии мы получаем возможность разлагать на множители вдвое более длинные числа, чем в предыдущем. Это позволяет построить 0-й.

С другой стороны, техника разложения на множители может достичь предела своих возможностей задолго до 2045. Хотя я думаю, что это маловероятно.

Не все согласятся с моими рекомендациями. NSA установило для своего Стандарта цифровой подписи (Digital Signature Standard, см. раздел 20.1) длину ключей от 512 до 1024 бит - намного меньше, чем я рекомендую для длительной безопасности. У Pretty Good Privacy ("Вполне надежный секрет", см. раздел 24.12) максимальная длина ключа RSA составляет 2047 бит. Аржан Ленстра, лучший в мире раскладыватель на множители, в течение последних 10 лет отказывается делать предсказания [949]. В -1-й приведены рекомендации Рона Ривеста для длины ключей, которые сделаны в 1990 году и кажутся мне слишком оптимистичными [1323]. Хотя его анализ на бумаге выглядит хорошо, в недавней истории можно найти примеры регулярно происходящих сюрпризов. Чтобы предохранить себя от последствия этих сюрпризов, есть смысл выбирать ключи с запасом.

Табл. 7-7.
Долгосрочный прогноз разложения на множители

Год	Длина ключа (в битах)
1995	1024
2005	2048
2015	4096
2025	8192
2035	16384
2045	32768

Минимальные оценки предполагают бюджет $25000, алгоритм "квадратичное решето" и скорость технического прогресса 20 процентов в год. Средние оценки предполагают бюджет 25 миллионов долларов, алгоритм "решето общего поля чисел" и скорость технического прогресса 33 процента в год. Максимальные оценки предполагают бюджет 25 миллиардов долларов, алгоритм "решето общего поля чисел", работающий со скоростью

решета специального поля чисел и скорость технического прогресса 45 процентов в год .

Всегда есть вероятность того, что успехи в разложении на множители будут поразительны и для меня, но я попытался учесть этот множитель в своих прогнозах . Но почему мне нужно верить? Я лишь продемонстрировал собственную глупость, занимаясь предсказаниями .

Табл. 7-8.
Оптимистичные рекомендации Ривеста для длины ключей (в битах)

Год	Минимальная	Средняя	Максимальная
1990	398	515	1289
1995	405	542	1399
2000	422	572	1512
2005	439	602	1628
2010	455	631	1754
2015	472	661	1884
2020	489	677	2017

Вычисление с помощью ДНК

Это похоже на волшебство. В 1994 году Леонард Эдлман (Leonard M. Adleman) продемонстрировал метод решения задачи **NP-полноты** (см. раздел 11.2) в биохимической лаборатории, используя молекулы ДНК для представления возможных решений задачи [17]. Задача, решенная Эдлманом, была частным случаем задачи направленного гамильтонова пути: дана карта городов, соединенных однонаправленными дорогами, нужно найти путь из города A в город Z, который проходит через каждый город на карте только один раз . Каждый город был представлен своей случайной цепочкой ДНК с 20 основаниями. С помощью обычных методов молекулярной биологии Эдлман синтезировал 50 пикомолей (30 миллионов миллионов молекул) цепочки ДНК, представляющей каждый город. Каждая дорога была представлена цепочкой ДНК с 20 основаниями, но эти цепочки выбирались не случайным образом: они умело выбирались так, чтобы "начало" цепочки ДНК, представляющей дорогу от города P к городу K ("Дорога PK") стремилась бы соединиться со цепочкой ДНК, представляющей город P, а конец Дороги PK стремился бы соединиться с городом K.

Эдлман синтезировал 50 пикомолей ДНК, представляющих каждую дорогу, смешал их вместе с ДНК гор одами, представляющей города, и добавил фермент лигазу, которая связывает вместе концы молекул ДНК. Правильно подобранная связь между цепочками ДНК для дорог и цепочками ДНК для городов приводит к тому, что лигаза соединяет цепочки ДНК для дорог вместе правильным образом . То есть, "Выход" дороги PK всегда будет соединен со "входом" какой-либо дороги, начинающейся в городе K, но никогда с "выходом" любой дороги или со "входом" дороги, которая начинается не в городе K. После тщательно ограниченного времени реакции лигаза создала большое количество цепочек ДНК, представляющих возможные, но все равно случайные объединения путей карты.

В этом супе из случайных путей Эдлман может найти малейший след - может быть единственной молекулы - ДНК, которая является ответом задачи. Используя обычные методы молекулярной биологии, он удалил все цепочки ДНК, представлявшие слишком короткие или слишком длинные пути . (Число дорог в нужном пути должно равняться числу городов минус один .) Затем он удалил все цепочки ДНК, которые не проходили через город A, затем те, которые шли мимо города B, и так далее. Молекула ДНК, прошедшая через это сито, и представляет собой нужную последовательность дорог, являясь решением задачи направленного гамильтонова пути .

По определению частный случай задачи **NP-полноты** может быть преобразован за полиномиальное время к частному случаю любой другой задачи **NP-полноты**, и, следовательно, к задаче о направленном гамильтоновом пути. С семидесятых годов криптологи пытались использовать задачи **NP-полноты** для криптографии с открытыми ключами.

Хотя частный случай, решенный Эдлманом, весьма прост (семь городов на карте, простым наблюдением задача моежт быть решена за несколько минут), это направление только начало развиваться, и не существует никаких препятствий для расширения данной методики на более сложные задачи . Таким образом, рассуждения о безопасности криптографических протоколов, основанных на задачах **NP-полноты**, рассуждения, которые до сих пор начинались словами, "Предположим, что у врага есть миллион процессоров, каждый из которых выполняет миллион проверок каждую секунду", скоро, может быть, будут начинаться словами, "Предположим, у врага есть тысяча ферментных ванн, емкостью по 20000 литров каждая ".

Квантовые вычисления

А теперь еще большая фантастика. В основе квантовых вычислений используется двойственная природа м атерии (и волна, и частица). Фотон может одновременно находиться в большом количестве состояний. Классич еским примером является то, что фотон ведет себя как волна, встречая частично прозрачное зеркало. Он одн овременно и отражается и проходит через зеркало подобно тому, как морская волна, ударяясь о волнолом с н ебольшим отверстием в нем, одновременно отразится от стены и пройдет сквозь нее . Однако, при измерении фотон ведет себя подобно частице, и только одно состояние может быть обнаружено .

В [1443] Питер Шор (Peter Shor) очертил принципы построения машины для разложения на множители, о сновanной на законах квантовой механики. В отличие от обычного компьютера, который можно представить как машину, имеющее в каждый момент времени единственное фиксированное состояние, квантовый компьютер обладает внутренней волновой функцией, являющейся суперпозицией комбинаций возможных основных с остояний. Вычисления преобразуют волновую функцию, меняя весь набор состояний единым действием . Таким образом, квантовый компьютер имеет преимущество над классическим конечным автоматом : он использует квантовые свойства для числа разложения на множители за полиномиальное время, теоретически позволяя взломать криптосистемы, основанные на разложении на множители или задаче дискретного логарифма .

Общепризнанно, что квантовый компьютер не противоречит фундаментальным законам квантовой механ ики. Однако, непохоже, что квантовая машина для разложения на множители будет построена в обозримом б удущем ... если вообще будет построена. Одним из главных препятствий является проблема некогерентности , которая является причиной потери отчетливости волновыми огибающими и приводит к сбою компьютера . Из-за некогерентности квантовый компьютер, работающий при 1К, будет сбиваться каждую наносекунду . Кроме того, для построения квантового устройства для разложения на множители потребуется огромное количество вент илей, а это может не дать построить машину. Для проекта Шора нужно совершенное устройство для возведения в степень. Внутренние часы не используются, поэтому для разложения на множители криптографически знач имых чисел могут потребоваться миллионы или, возможно, миллиарды индивидуальных вентилей . Если мини мальная вероятность отказа каждого из n квантовых вентилей равна p, то среднее количество испытаний, нео бходимое для достижения успеха, составит $(1/(1-p))^n$. Число нужных вентилей, по видимому, растет полином иально с ростом длины числа (в битах), поэтому число требуемых попыток будет расти с увеличением длины используемых чисел сверхэкспоненциально - хуже чем при разложении делением !

Поэтому, хотя квантовое разложение на множители вызывает восхищение в академических кругах, малов ероятно, что оно будет иметь практическое значение в обозримом будущем . Но не говорите потом, что я вас не предупреждал.

7.3 Сравнение длин симметричных и открытых ключей

Система взламывается обычно в ее слабейшем месте. Если вы проектируете систему, которая использует и симметричную криптографию, и криптографию с открытыми ключами, то длины ключей для криптографии каждого типа должны выбираться так, чтобы вскрыть любой из компонентов системы было одинаково трудно . Бессмысленно использовать симметричный алгоритм со 128-битовым ключом вместе с алгоритмом с открыт ыми ключами, использующим 386-битовый ключ. Точно так же бессмысленно использовать в одной системе симметричный алгоритм с 56-битовым ключом и алгоритм с открытыми ключами, применяющий 1024-битовый ключ.

В -2-й перечислены длины модулей открытых ключей, трудность разложения которых на множители сра внима со сложностью вскрытием грубой силой сопоставленных длин популярных симметричных ключей .

Табл. 7-9.
Длины симметричных и открытых ключей с аналогичной устойчивостью к вскрытию грубой силой

Длина симметричного ключа (в битах)	Длина открытого ключа (в битах)
56	384
64	512
80	768
112	1792
128	2304

Из этой таблица можно сделать вывод, что если вы достаточно беспокоитесь о своей безопасности, чтобы выбрать симметричный алгоритм со 112-битовым ключом, вам следует выбрать длину модуля в вашем алгоритме с открытыми ключами порядка 1792 бит. Однако, в общем случае следует выбирать длину открытого ключа более безопасную, чем длина вашего симметричного ключа. Открытые ключи обычно используются дольше и применяются для защиты большего количества информации.

7.4 Вскрытие в день рождения против однонаправленных хэш-функций

Существует два способа вскрытия однонаправленных хэш-функций грубой силой. Первый наиболее очевиден: дано значение хэш-функции сообщения, *H(M)*, врагу хотелось бы суметь создать другой документ, *M'*, такой, что *H(M')=H(M)*. Второй способ более тонок: врагу хотелось бы найти два случайных сообщения, *M* и *M'*, таких, что *H(M')=H(M)*. Такой способ называется **столкновением** и является более простым, чем первый, способом вскрытия.

Парадокс дня рождения является стандартной статистической проблемой. Сколько человек должно собраться в одной комнате, чтобы с вероятностью 1/2 хотя бы у кого-нибудь из них был бы общий с вами день рождения? Ответ - 183. Хорошо, а сколько людей должно собраться, чтобы с вероятностью 1/2 хотя бы у двоих из них был бы общий день рождения? Ответ удивителен - 23. 23 человека, находящихся в комнате, образуют 253 различных *пары*.

Найти кого-нибудь с тем же днем рождения - аналогия с первым способом вскрытия, найти двух человек с произвольным одинаковым днем рождения - аналогия со вторым способом. Второй способ широко известен как **вскрытие в день рождения**.

Предположим, что однонаправленная хэш-функция безопасна, и лучшим способом ее вскрытия является вскрытие грубой силой. Результатом функции является *m*-битовое число. Поиск сообщения, хэш-значение которого совпадает с заданным, в среднем потребовал бы хэширования 2^m случайных сообщений. А для обнаружения двух сообщений с одинаковым хэш-значением потребуется только $2^{m/2}$ случайных сообщений. Компьютеру, который хэширует миллион сообщений в секунду, потребовалось бы 600000 лет, чтобы найти второе сообщение с тем же 64-битовым хэш-значением. Тот же компьютер сможет найти пару сообщений с общим хэш-значением примерно за час

Это значит, что, если вы опасаетесь вскрытия в день рождения, вы должны выбирать длину хэш-значения в два раза длиннее, чем вам потребовалось бы в противном случае. Например, если вы хотите уменьшить вероятность взлома вашей системы до 1 шанса из 2^{80}, используйте 160-битовую однонаправленную хэш-функцию.

7.5 Каков должны быть длина ключа?

На этот вопрос нет единого ответа, ответ этот зависит от ситуации. Чтобы понять, какая степень безопасности вам нужна, вы должны задать себе несколько вопросов. Сколько стоит ваша информация? Как долго она должна безопасно храниться? Каковы ресурсы ваших врагов?

Список клиентов может стоить $1000. Финансовая информация при неожиданном разводе могла бы стоить $10000. Реклама и данные маркетинга для большой корпорации могли бы стоить 1 миллион долларов. Главный ключ для системы электронных наличных может стоить миллиарды.

В мире торговли предметами потребления секреты должны только сохраняться в течение нескольких минут. В газетном бизнесе сегодняшние секреты - это завтрашние заголовки. Информация о разработке какого-то продукта, возможно, должна будет храниться в секрете в течение года или двух Изделия(программы) могла бы была бы должна остаться секретом в течение года или два. Данные переписи США в соответствии с законом должны храниться в секрете в течение 100 лет.

Список гостей, приглашенных на вечер-сюрприз в честь дня рождения вашей сестры, интересен только в ашим любопытным родственникам. Торговые секреты корпорации представляют интерес для конкурирующих компаний. Военные секреты интересны вражеским военным.

В этих терминах даже можно определить требования к безопасности Можно. Например:

> Длина ключа должна быть такой, чтобы взломщик, готовый потратить 100 миллионов долларов, мог взломать систему в течение года с вероятностью не более, чем $1/2^{32}$, даже с учетом скорости технического прогресса 30 процентов в год.

В -3-й, частично взятой из [150], приведены оценки требований к безопасности для различной информ ации:

Будущую вычислительную мощь оценить нелегко, но вот разумное эмпирическое правило: эффективность вычислительных средств удваивается каждые 18 месяцев и возрастает на порядок каждые 5 лет. Следовательно, через 50 лет самые быстрые компьютеры будут в 10 миллиардов быстрее, чем сегодня! Кроме того, не забывайте, что эти числа относятся только к универсальным компьютерам, кто знает, какие специализированные ус тройства для вскрытия криптосистем будут разработаны в следующие 50 лет?

Предполагая, что криптографический алгоритм будет использоваться в ближайшие 30 лет, вы можете пре д-ставить себе, насколько он должен быть безопасен. Алгоритм, созданный сегодня, возможно не станет широко использоваться до 2000 года, и все еще будет использоваться в 2025 для шифрования сообщений, которые должны оставаться в секрете до 2075 года и позже.

Табл. 7-10.
Требования к безопасности различной информации

Типы трафика	Время жизни	Минимальная длина ключа (в битах)
Тактическая военная информация	минуты/часы	56-64
Объявления о продуктах, слиянии компаний, процен т-ных ставках	дни/недели	64
Долговременны бизнес-планы	годы	64
Торговые секреты (например, рецепт кока-колы)	десятилетия	112
Секреты водородной бомбы	>40 лет	128
Личности шпионов	>50 лет	128
Личные дела	>50 лет	128
Дипломатические конфликты	>65 лет	128
Данные переписи США	100 лет	по меньшей мере 128

7.6 Caveat emptor[1]

Вся эта глава - просто много чепухи. This entire chapter is a whole lot of nonsense. Смешно говорить даже о самом понятии предсказания вычислительной мощи на 10, а тем более на 50 лет вперед. Эти расчеты приведены только для ориентировки, ни для чего больше. Экстраполируя прошлое, мы получаем будущее, которое, возможно, будет иметь мало общего с грядущей реальностью.

Будьте консерваторами. Если ваши ключи длиннее, чем вам кажется необходимым, то меньшее количество технологических сюрпризов сможет повредить вам.

[1] Да будет осмотрителен покупатель (латин.)

Глава 8
Управление ключами

У Алисы и Боба есть безопасная система связи. Они играют в мысленный покер, одновременно подписыв аю т контракты и даже меняют цифровые наличные. Их протоколы безопасны. Их алгоритмы -самые лучшие. К несчастью, они покупают свои ключи от "Keys-R-Us" Евы, чей лозунг - "Вы можете доверять нам: Безопасность - среднее имя человека, которого туристический агент нашей бывшей тещи встретил в "Kwik-E-Mart".

Ева не нужно вскрывать алгоритмы. Ей не нужно полагаться на тонкие дефекты протоколов. Она может и с пользовать их ключи для чтения всех сообщений Алисы и Боба, не прикладывая никаких криптоаналитических усилий.

В реальном мире управление ключами представляет собой самую трудную часть криптографии. Проектир о вать безопасные криптографические алгоритмы и протоколы не просто, но Вы можете положиться на большой объем академических исследований. Сохранить секрет ключей намного труднее.

Криптоаналитики часто вскрывают и симметричные криптосистемы, и криптосистемы с открытыми ключ а ми через распределение ключей. Зачем Еве беспокоиться об проблеме вскрытия криптографического алгоритма целиком, если она может восстановить ключ из-за неаккуратного хранения ключа? Зачем ей тратить 10 ми л лионов долларов на создание машина для криптоанализа, если она может подкупить клерка за 1000 долларов? Миллион долларов за клерка связи на хорошем месте в дипломатическом посольстве может быть выгодной сделкой. Уолкеры годами продавали Советам ключи шифрования ВМС США. Руководитель контрразведки ЦРУ стоил меньше 2 миллионов долларов, включая жену. Это намного дешевле, чем строить огромные маш и ны вскрытия и нанимать блестящих криптоаналитиков. Ева может выкрасть ключи. Она может арестовать или похищать кого-то, кто знает ключи. Она может совращать кого-то и получать ключи таким образом. (Морские пехотинцы, охранявшие посольство США в Москве не устояли перед подобной атакой.) Намного проще нах о дить дефекты в людях, чем в криптосист емах.

Алиса и Боб должны защищать и свой ключ, и в той степени шифруемые им данные. Если ключ не изменять регулярно, то количество данных может быть огромно. К сожалению, многие коммерческие продукты просто объявляют "Мы используем DES" и забывают обо всем остальном. Результаты не слишком впечатляют.

Например, программа DiskLock для Macintosh (версия 2.l), продававшаяся в большинстве магазинов пр о граммного обеспечения, претендует на безопасное шифрование DES. Она шифрует файлы, используя DES. Ре а лизация DES алгоритма правильна. Однако, DiskLock сохраняет ключ DES вместе с зашифрованным файлом. Если вы знаете, где искать ключ, и хотеть прочитать файл, шифрованный DiskLock с помощью DES, восстан о вите ключ из шифрованного файла и затем расшифровывать файл. Не имеет значение, что программа использ у ет шифрование DES - реализация абсолютно небезопасна.

Дальнейшую информацию относительно управления ключами можно найти в [457, 98, 1273, 1225, 775, 357]. В следующих разделах обсуждаются некоторые из вопросов и решений.

8.1 Генерация ключей

The security of an algorithm rests in the key. If you're using a cryptographically weak process to generate keys, then your whole system is weak. Eve need not cryptanalyze your encryption algorithm; she can cryptanalyze your key generation algorithm.

Безопасность алгоритма сосредоточена в ключе. Если вы используете криптографически слабый процесс для генерации ключей, то ваша система в целом слаба. Еве не нужно криптоанализировать ваш алгоритм шифров а ния, она может криптоанализировать ваш алгоритм генерации ключей.

Уменьшенные пространства ключей

DES использует 56-битовый ключ с битами. Любая правильно заданная 56-битовая строка может быть кл ю чом, существует 2^{56} (10^{16}) возможных ключей. Norton Discreet for MS-DOS (версии 8.0 и более ранние) разр е шает пользоваться только ключам ASCII, делая старший бит каждого байта нолем. Программа также преобр а зует символы нижнего регистра в верхний регистр (так что пятый бит каждого байта всегда противоположен шестому бита) и игнорирует бит младшего разряда каждого байта, что приводит к пространству в 2^{40} возможных ключей. Эти ущербные процедуры генерации ключей сделали свою реализацию DES в десять тысяч раз проще для вскрытия.

7-й содержит число возможных ключей для различных ограничений на входные строки. В 6-й приведено время, требуемое для исчерпывающего перебора всех возможных ключей при миллионе попыток в секу нду.

Могут быть использованы для вскрытия грубой силой любые специализированные аппаратные и параллел ь-

ные реализации. При проверке миллиона ключей в секунду (одной машиной или несколькими параллельно) физически возможно расколоть ключи из символов нижнего регистра и ключи из цифр и символов нижнего регистра длиной до 8 байтов, алфавитно-цифровые ключи - длиной до 7 байтов, ключи из печатаемых символов и ASCII-символов - длиной до 6 байтов, в ключи из 8-битовых ASCII-символов - длиной до 5 байтов.

Табл. 8-1.
Количество возможных ключей в различных пространствах ключей

	4 байта	5 байтов	6 байтов	7 байтов	8 байтов
Строчные буквы (26)	460000	$1.2*10^{7}$	$3.1*10^{8}$	$8.0*10^{9}$	$2.1*10^{11}$
Строчные буквы и цифры (36)	1700000	$6.0*10^{7}$	$2.2*10^{9}$	$7.8*10^{10}$	$2.8*10^{12}$
Алфавитные и цифровые символы (62)	$1.5*10^{7}$	$9.2*10^{8}$	$5.7*10^{10}$	$3.5*10^{12}$	$2.2*10^{14}$
Печатаемые символы (95)	$8.1*10^{7}$	$7.7*10^{9}$	$7.4*10^{11}$	$7.0*10^{13}$	$6.6*10^{15}$
Символы ASCII (128)	$2.7*10^{8}$	$3.4*10^{10}$	$4.4*10^{12}$	$5.6*10^{14}$	$7.2*10^{16}$
8-битовые ASCII символы (256)	$4.3*10^{9}$	$1.1*10^{12}$	$2.8*10^{14}$	$7.2*10^{16}$	$1.8*10^{19}$

Табл. 8-2.
Время исчерпывающего поиска различных пространства ключей (при одном миллионе проверок в секунду)

	4 байта	5 байтов	6 байтов	7 байтов	8 байтов
Строчные буквы (26)	0.5 секунды	12 секунд	5 минут	2.2 часа	2.4 дня
Строчные буквы и цифры (36)	1.7 секунды	1 минута	36 минут	22 часа	33 дня
Алфавитные и цифровые символы (62)	15 секунд	15 минут	16 часов	41 день	6.9 года
Печатаемые символы (95)	1.4 минуты	2.1 часа	8.5 дня	2.2 года	210 лет
Символы ASCII (128)	4.5 минуты	9.5 часа	51 день	18 лет	2300 лет
8-битовые ASCII символы (256)	1.2 часа	13 дней	8.9 года	2300 лет	580000 лет

И помните, вычислительная мощь удваивается каждые 18 месяцев. Если вы хотите, чтобы ваши ключи были устойчивы к вскрытию грубой силой в течение 10 лет, вы должны соответствующим образом планировать и с-пользование ключей.

Обедненный выбор ключей

Когда люди сами выбирают ключи, они выбирают ущербные ключи. Они с большей вероятностью выберут "Barney", чем "*9 (hH/A". Это не всегда происходит из-за плохой практики, просто "Barney" легче запомнить чем "*9 (hH/A". Самый безопасный алгоритм в мире не сильно поможет, если пользователи по привычке выб-ирают имена своих жен (мужей) для ключей или пишут свои ключи на небольших листочках в бумажниках. И н-теллектуальное вскрытие грубой силой не перебирает все возможные ключи в числовом порядке, но пробует сначала очевидные ключи.

Это называется **вскрытием со словарем**, потому что нападющий использует словарь общих ключей. Дэн-ел Кляйн (Daniel Klein) смог расколоть 40 процентов паролей на среднем компьютере, используя этот способ вскрытия [847, 848]. Нет, он не перебирал один пароль за другим, пытаясь войти в систему. Он скопировал з-ашифрованный файл паролей и предпринял вскрытие автономно. Вот, что он пробовал:

1. В качестве возможного пароля имя пользователя, инициалы, имя бюджета и другую связанную с ч-еловеком информацию. В целом, на основе такой информации пробовалось до 130 различных паролей. Вот некоторые из паролей, проверявшихся для имени бюджета **klone** и пользователя "Daniel V. Klein": klone, klone0, klonel, klonel23, dvk, dvkdvk, dklein, Dklein, leinad, nielk, dvklein, danielk, DvkkvD, DANIEL-KLEIN, (klone), KleinD, и так далее.
2. Слова из различных баз данных. Использовались списки мужских и женских имен (всего около 16000), названия мест (включая изменения, поэтому рассматривались и "spain", " Spanish", и "Spaniard"), имена известных людей, мультфильмы и мультипликационные герои, заголовки, герои и места из фильмов и научной фантастики, мифические существа (добытые из *Bullfinch's Mythology* и

словарей мифических животных), спорт (включая названия команд, прозвища и специальные терм ины), числа (записанные как цифрами - '2001", так и буквами " twelve"), строки символов и чисел ("a", "aa", "aaa", "aaaa" и т.д.), китайские слоги (из Piny in Romanization of Chinese, международного стан дарта письма по китайски на англоязычной клавиатуре), Библия короля Джеймса; биологические те рмины, разговорные и вульгарные выражения (типа "fuckyou", "ibmsux" и "deadhead"), стандарты кл авиатуры (типа "qwerty", "asdf" и "zxcvbn"), сокращения (типа "roygbiv" - первые буквы названий цв етов радуги по английски - и "ooottafagvah" - мнемоническая схема для запоминания 12 черепных не рвов), имена компьютеров (полученные из /etc/hosts), герои, пьесы и места действия у Шекспира, с амые распространенные слова языка Идиш, названия астероидов, совокупность слов из различных те хнических статей, опубликованных ранее Кляйном. Итого, для пользователя рассматривалось более чем 60000 отдельных слов (с отбрасыванием дубликатов в различных словарях).

3. Вариации слов из пункта 2. Это включало перевод первого символа в верхний регистр или его замену управляющим символом, перевод всего слова в верхний регистр, инверсию регистра слова (с и без вышеупомянутого изменения регистра первой буквы), замену буквы "o" на цифру "0" (так, чтобы сл ово "scholar" было также проверено как "sch0lar"), замену буквы "l" на цифру "1" (так, чтобы слово "scholar" было бы также проверено как "scho1ar") и выполнение аналогичных манипуляций с буквой "z" и цифрой "2", а также с буквой "s" и цифрой "5". Другая проверка состояла из перевода слова во множественное число (независимо от того, было ли слово существительным) с учетом необходимых правил, чтобы "dress" заменилось на "dresses", "house" - на "houses", а "daisy" - на "daisies". Хотя Кляйн не жестко придерживался правил преобразования ко множественному числу, поэтому "datum" стала "datums" (а не "data"), "sphynx" - "sphynxs" (а не "sphynges"). Аналогично, для преобразования слов добавлялись суфиксы "-ed", "-er" и "-ing", подобно "phase" в "phased," "phaser" и "phasing". Эти дополнительные проверки добавили еще 1000000 слов к списку возможных паролей, которые пров ерялись для каждого пользователя.

4. Различные варианты преобразования к верхнему регистру слов пункта 2, не рассматривавшихся в пункте 3. Сюда вошло преобразование к верхнему регистру одиночных символов (так, чтобы "michael" был также проверен как "m Ichael", "miChael", "micHael", "michAel", и т.д.), преобразование к верхнему регистру пары символов ("MIchael", "MiChael", "MicHael", ..., "mIChael", "mIcHael", и т.д.), преобразование к верхнему регистру трех символов, и т.д. Изменения одиночного символа доб авили к проверяемым примерно еще 400000 слов, а изменения пары символов - 1500000 слов. Измен ения трех символов добавляли по крайней мере еще 3000000 слов для каждого пользователя, если для завершения тестирования хватало времени. Проверка изменения четырех, пяти и шести символов была признана непрактичной, так как для их проверки не хватало вычислительных мощностей.

5. 5. Иностранные слова для иностранных пользователей. Специфический тест, который был выполнен, проверял пароли из китайского языка для пользователей с китайскими именами. Для китайских сл огов использовался стандарт Pinyin Romanization, слоги объединялись вместе в одно-, двух- и тре хсложные слова. Так как не было выполнено предварительной проверки слов на значимость, использ овался исчерпывающий перебор. Так как в системе Pinyin существует 298 китайских слогов, то имеется 158404 слов с двумя слогами, и немного больше 16000000 слов с тремя слогами. Подобный способ вскрытия мог бы быть легко использован и для английского языка, с учетом правил образования пр оизносимых ничего не значащих слов.

6. Пары слов. Объем такого исчерпывающего теста колеблется. Чтобы упростить тест, из */usr/dict/words* использовались только слова длиной три или четыре символа. Даже при этом, число пар слов сост авило приблизительно десять миллионов.

Вскрытие со словарем намного мощнее, когда оно используется против файла ключей, а не против одного ключа. Одиночный пользователь может быть достаточно разумен и выбрать хорошие ключи. Если из тысячи людей каждый выбирает собственный ключ как пароль компьютерной системы, то велика вероятность того, что по крайней мере один человек выберет ключ, имеющийся в словаре взломщика.

Случайные ключи

Хорошими ключами являются строки случайных битов, созданные некоторым автоматическим процессом. Если длина ключа составляет 64 бита, то все возможные 64-битовые ключи должны быть равновероятны. Ген ерируйте биты ключей, пользуясь либо надежным источником случайных чисел (см. раздел 17.14), либо крипт ографически безопасным генератором псевдослучайных битов (см. главы 16 и 17.) Если такие автоматические процессы недоступны, бросайте монетку или кости.

Это важно, но не увлекайтесь обсуждением того, является ли шум из звуковых источников более случайным, чем шум из радиоактивного распада. Ни один из этих источников случайного шума не совершенен, но все они, скорее всего, будут достаточно хороши. Для генерации ключей важно использовать хороший генератор случа йных чисел, но гораздо важнее использовать хорошие алгоритмы шифрования и процедуры управления ключ а-

ми. Если вы беспокоитесь о случайности ваших ключей, используйте описанную ниже методику перемалывания ключа.

Некоторые алгоритмы шифрования имеют слабые ключи - специфические ключи, менее безопасные чем другие ключи. Я советую проверять слабость ключа ключей и, обнаружив ее, генерировать новый. У DES только 16 слабых ключей в пространстве 2^{56}, так что вероятность получить один из этих ключей невероятно мала. Заявлялось, что криптоаналитик не будет знать о том, что используется слабый ключ, и, следовательно, не сможет получить никакой выгоды из их случайного использования. Также заявлялось, что информацию криптоаналитику дает совсем не использование слабых ключей. Однако, проверка немногих слабых ключей настолько проста, что кажется глупым пренебречь ею.

Генерация ключей для систем криптографии с открытыми ключами тяжелее, потому что часто ключи должны обладать определенными математическими свойствами (возможно, они должны быть простыми числами, квадратичным остатком, и т.д.). Методы генерации больших случайных простых чисел рассматриваются в разделе 11.5. Важно помнить, что с точки зрения управления ключами случайные стартовые последовательности для таких генераторов должны быть действительно случайны.

Генерация случайного ключа возможна не всегда. Иногда вам нужно помнить ваш ключ. (Интересно, сколько времени вам понадобится, чтобы запомнить 25e8 56f2 e8ba c820?). Если вам надо генерировать простой для запоминания ключ, замаскируйте его. Идеалом является то, что легко запомнить, но трудно угадать. Вот несколько предложений:

— Пары слов, разделенные символом пунктуации, например, " turtle*moose" или "zorch!splat"

— Строки букв, являющиеся акронимами длинных фраз, например, "Mein Luftkissenfahrzeug ist voller Aale!" служит для запоминания ключа "MLivA!"

Ключевые фразы

Лучшим решением является использование вместо слова целой фразы и преобразование этой фразы в ключ. Такие фразы называются **ключевыми фразами**. Методика с названием **перемалывание ключа** преобразует легко запоминающиеся фразы в случайные ключи. Для преобразования текстовой строки произвольной длины в строку псевдослучайных бит используте однонаправленную хэш-функцию. Например, легко запоминающаяся текстовая строка:

```
My name is Ozymandias, king of kings. Look on my works, ye mighty, and despair. [1]
```

может "перемолоться" в такой 64-битовый ключ:

```
e6c1 4398 5ae9 0a9b
```

Конечно, может быть нелегко ввести в компьютер целую фразу, если вводимые символы не отображаются на экране. Разумные предложения по решению этой проблемы будут оценены.

Если фраза достаточно длинна, то полученный ключ будет случаен. Вопрос о точном смысле выражения "достаточно длинна" остается открытым. Теория информации утверждает, что информационная значимость стандартного английского языка составляет около 1.3 бита на символ (см. раздел 11.1). Для 64-битового ключа достаточной будет ключевая фраза, состоящая примерно из 49 символов, или 10 обычных английских слов. В качестве эмпирического правила используйте пять слов для каждых 4 байтов ключа. Это предложение работает с запасом, ведь в нем не учитываются регистр, пробелы и знаки пунктуации.

Этот метод также можно использовать для генерации закрытых ключей в криптографических системах с открытыми ключами: текстовая строка преобразуется в случайную стартовую последовательность, а эта последовательность может быть использована в детерминированной системе, генерирующей пары открытый ключ/закрытый ключ.

Выбирая ключевую фразу, используйте что-нибудь уникальное и легко запоминающееся. Не выбирайте фразы из книг - пример с "Ozymandias" в этом смысле плох. Легко доступны и могут быть использованы для вскрытия со словарем и собрание сочинений Шекспира, и диалоги из *Звездных войн*. Выберите что-нибудь туманное и личное. Не забудьте о пунктуации и преобразовании регистра, если возможно включите числа и неалфавитные символы. Плохой или искаженный английский, или даже любой иностранный язык, делает ключевую фразу более устойчивой к вскрытию со словарем. Одним из предложений является использование фразы, которая является "потрясающей ерундой", чем-то таким, что вы вряд ли запомните и вряд ли запишете.

Несмотря на все написанное здесь маскировка не заменяет истинную случайность. Лучшими являются случайные ключи, которые так тяжело запомнить.

---

[1] Я Озимандиас, царь царей. Вы, сильные мира сего, смотрите на мои труды и трепещите.

Стандарт генерации ключей X9.17

Стандарт ANSI X9.17 определяет способ Генерации ключей (см. 7th) [55]. Он не создает легко запоминающиеся ключи, и больше подходит для генерации сеансовых ключей или псевдослучайных чисел в системе . Для генерации ключей используется криптографический алгоритм DES, но он может быть легко заменен любым другим алгоритмом.

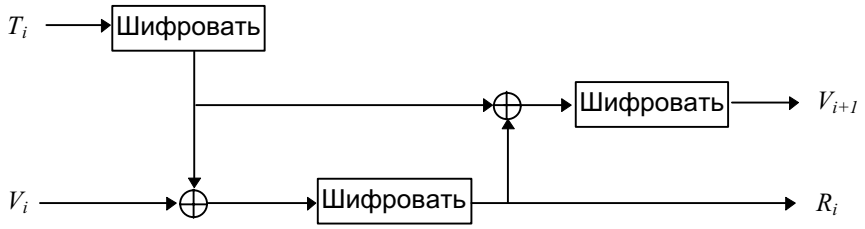


Рис. 8-1. Генерация ключей ANSI X9.17

Пусть $E_K(X)$ - это X, зашифрованный DES ключом K, специальным ключом, предусмотренным для генерации секретных ключей. V_0 - это секретная 64-битовая стартовая последовательность. T - это метка времени. Для генерации случайного ключа R_i вычислим:

$R_i = E_K(E_K(T_i) \oplus V_i)$

Для генерации V_{i+1}, вычислим:

$V_{i+1} = E_K(E_K(T_i) \oplus R_i)$

Для превращения R_i в ключ DES, просто удалите каждый восьмой бит. Если вам нужен 64-битовый ключ, используйте ключ без изменения. Если вам нужен 128-битовый ключ, создайте пару ключей и объедините их.

Генерация ключей в министерстве обороны США

Министерство обороны США для генерации случайных ключей рекомендует использовать DES в режиме OFB (см. раздел 9.8) [1144]. Создавайте ключи DES, используя системные вектора прерывания, регистры состояния системы и системные счетчики. Создавайте вектор инициализации, используя системные часы, идентификатор системы, с также дату и время. Для открытого текста используйте 64-битовые величины, созданные кем-то другим, например, 8 символов, введенных системным администратором . Используйте в качестве своего ключа результат.

8.2 Нелинейные пространства ключей

Вообразите, что вы - это военная криптографическая организация, создающая криптографический модуль для ваших войск. Вы хотите использовать безопасный алгоритм, но что будет, если аппаратура попадет во вражеские руки? Ведь вы не хотите, чтобы ваши приборы использовались для защиты *вражеских* секретов.

Если вы можете поместить ваш алгоритм в защищенный модуль, то вот, что вы можете сделать. Потребуйте, чтобы модуль правильно работал только с ключами специальной и секретной формы, а со всеми другими ключами для шифрования использовался сильно ослабленный алгоритм. Можно сделать так, чтобы вероятность того, что кто-то, не знающий этой специальной формы, случайно наткнется на правильный ключ, была исчезающе малой.

Получившееся пространство ключей называется **нелинейным**, потому что ключи не являются одинаково сильными. (Противоположным является линейное, или плоское, пространство ключей.) Простым способом добиться этого можно, создавая ключ, состоящий из двух частей: непосредственно ключа и некоторой фиксированной строки, шифрованной этим ключом. Модуль расшифровывает строку, используя ключ. Если результатом оказывается фиксированная строка, то ключ используется как обычно, если нет, то используется другой, слабый алгоритм. Если алгоритм имеет 128-битовый ключ и 64-битовый размер блока, то длина полного ключа - 192 бита. Таким образом, у алгоритма 2^{128} эффективных ключа, но вероятность случайно выбрать правильный составляет один шанс из 2^{64}.

Вы можете сделать еще хитрее. Можно разработать такой алгоритм, что некоторые ключи будут сильнее других. У алгоритма не будет слабых ключей - ключей, которые с очевидностью являются недостаточно защищенными - и тем не менее у него будет нелинейное пространство ключей.

Это работает только, если используется секретный алгоритм, который враг не может перепроектировать, или если различие в силе ключей достаточно тонко, чтобы враг не смог о нем догадаться. NSA проделывало это с секретными алгоритмами в своих модулях Overtake (см. раздел 25.1). Делали ли они то же самое с Skipjack (см. раздел 13.12)? Неизвестно.

8.3 Передача ключей

Алиса и Боб собираются для безопасной связи использовать симметричный криптографический алгоритм, им нужен общий ключ. Алиса генерирует ключ, используя генератор случайных ключей. Теперь она должна безопасно передать его Бобу. Если Алиса сможет где-то встретить Боба (какие-нибудь задворки, комната без окон или одна из лун Юпитера), то она сможет передать ему копию ключа. В противном случае, у них есть пр облема. Криптография с открытыми ключами решает проблему легко и с минимумом предварительных соглаш ений, но эти методы не всегда доступны (см. раздел 3.1). Некоторые системы используют альтернативные кан алы, считающиеся безопасными. Алиса могла бы посылать Бобу ключ с доверенным посыльным. Она могла бы послать ключ заказной почтой или ночной службой доставки. Она могла бы устанавливать другой канал связи с Бобом и надеяться, что его то никто не подслушивает.

Алиса могла бы послать Бобу симметричный ключ по их каналу связи - тот, который они собираются ши фровать. Но глупо передавать ключ шифрования канала по этому же каналу в открытом виде, кто-то, подслуш ивающий канал, наверняка сможет расшифр овывать все сообщения.

Стандарт X9.17 [55] определяет два типа ключей: **ключи шифрования ключей** и **ключи данных**. Ключами шифрования ключей при распределении шифруются другие ключи. Ключи данных шифруют сами сообщ ения. Ключи шифрования ключей должны распределяться вручную, (хотя они могут быть в безопасности в з ащищенном от взлома устройстве, таком как кредитная карточка), но достаточно редко. Ключи данных распр еделяются гораздо чаще. Подробности можно найти в [75]. Эта идей двухсвязных ключей часто используется при распределении ключей.

Другим решением проблемы распределения является разбиение ключа на несколько различных частей (см. раздел 3.6) и передача их по различным каналам. Одна часть может быть послана телефоном, другая - почтой, третья - службой ночной доставки, четвертая - почтовым голубем, и так далее, (см. 6-й). Так противник может собрать все части, кроме одной, и все равно ничего не узнает про ключ. Этот метод будет работать во всех сл учаях, кроме крайних. В разделе 3.6 обсуждаются схемы разбиения ключа на несколько частей. Алиса могла бы даже применить схему совместно используемого секрета, (см. раздел 3.7), что даст возможность Бобу восст анавливать ключ, если некоторые из частей потеряны при перед аче.

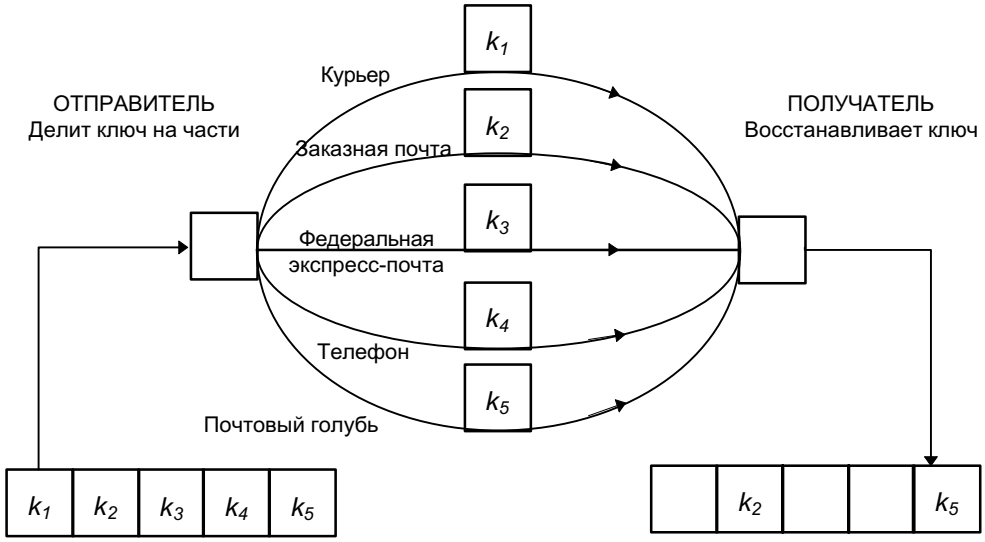


Рис. 8-2. Распределение ключей по параллельным каналам.

Алиса безопасно передает Бобу ключ шифрования ключей или при личной встрече, или с помощью только что рассмотренной методики разбиения. Как только и у Алисы, и у Боба будет ключ шифрования ключей, Ал иса сможет посылать Бобу ключи данных на день по тому же самому каналу связи, шифруя при этом каждый ключ данных ключом шифрования ключей. Так как трафика, шифруемый ключом шифрования ключей незн ачителен, то этот ключ часто менять не нужно. Однако, так как компрометация ключа шифрования ключей м ожет скомпрометировать все сообщения, шифрованное использованными ключами данных, которые были з ашифрован этим ключом шифрования ключей, этот ключ должен храниться в безопасности.

Распределение ключей в больших сетях

Ключи шифрования ключей, общие для пары пользователей, хорошо использовать в небольших сетях, но с увеличением сети такая система быстро становится громоздкой. Так как каждая пара пользователей должна

обменяться ключами, общее число обменов ключами в сети из n человек равно n(n - l)/2.

В сети с шестью пользователями потребуется 15 обменов ключами. В сети из 1000 пользователей понадобится уже около 500000 обменов ключами. В этих случаях работа сети гораздо более эффективна при использовании центрального сервера (или серверов) ключей.

Кроме того, любой из протоколов симметричной криптографии или криптографии с открытыми ключами, приведенных в разделе 3.1, подходит для безопасного распределения ключей.

8.4 Проверка ключей

Как Боб узнает, получив ключ, что ключ передан Алисой, а не кем-то другим, кто выдает себя за Алису? Все просто, если Алиса передает ему ключ при личной встрече. Если Алиса посылает свой ключ через доверенного курьера, то курьеру должен доверять и Боб. Если ключ зашифрован ключом шифрования ключей, то Боб должен доверять тому, что этот ключ шифрования ключей есть только у Алисы. Если для подписи ключа Алиса использует протокол электронной подписи, Боб при проверке подписи должен доверять базе данных открытых ключей,. (Ему также придется считать, что Алиса сохранила свой ключ в безопасности.) Если Центр распределения ключей (Key Distribution Center, KDC) подписывает открытый ключ Алисы, Боб должен считать, что его копия открытого ключа KDC не была подменена.

Наконец, тот, кто управляет всей сетью вокруг Боба, может заставить его думать все, что ему хочется. Мэллори может послать шифрованное и подписанное сообщение, выдавая себя за Алису. Когда Боб, проверяя подпись Алисы, обратится к базе данных открытых ключей, Мэллори может возвратить ему собственный открытый ключ. Мэллори может создать свой собственный поддельный KDC и подменить открытый ключ настоящего KDC ключом своего собственного изделия. Боб никак не сможет это обнаружить.

Некоторые люди использовали этот аргумент, утверждая, что криптография с открытыми ключами бесполезна. Так как единственный способ Алисе и Бобу знать наверняка, что никто не взломал их ключи, - это личная встреча, то криптография с открытыми ключами вообще не обеспечивает безопасность.

Эта точка зрения наивна. Теоретически все правильно, но действительность гораздо сложнее. Криптография с открытыми ключами, используемая вместе с электронными подписями и надежными KDC, сильно усложняет подмену одним ключом другого. Боб никогда не может быть абсолютно уверен, что Мэллори не контролирует его реальность полностью, но Боб может знать наверняка, что такая подмена реальности потребует гораздо больше ресурсов, чем сможет заполучить реальный Мэллори.

Боб мог бы также проверять ключ Алисы по телефону, получив возможность услышать ее голос. Распознавание голоса действительно является хорошей схемой идентификации личности. Если речь идет об открытом ключе, он может безопасно его повторить его даже при угрозе подслушивания. Если это секретный ключ, он может использовать для проверки ключа одностороннюю хэш-функцию. Оба TSD PGP (см. раздел 24.12.) и AT$T (см. Раздел 24.18) используют этот способ проверки ключей.

Иногда может даже не важно точно проверять, кому принадлежит открытый ключ. Может понадобиться проверить, что он принадлежит тому же человеку, что и год назад. Если кто-то посылает банку подписанное сообщение о переводе денег, банк волнует не то, кто конкретно снимает деньги, а только то, чтобы этот человек был тем, кто внес деньги в первый раз.

Обнаружение ошибок при передаче ключей

Иногда ключи искажаются при передаче. Это является проблемой, так как искаженный ключ может привести к мегабайтам нерасшифрованного шифротекста. Все ключи должны передаваться с обнаружением ошибок и исправлением битов. Таким образом ошибки при передаче могут быть легко обнаружены и, если потребуется, ключ может быть послан еще раз.

Одним из наиболее широко используемых методов является шифрование ключом некоторой постоянной величины и передача первых 2-4 байт этого шифротекста вместе с ключом. У получателя делается то же самое. Если шифрованные константы совпадают, то ключ был передан без ошибки. Вероятность ошибки находится в диапазоне от $1/2^{16}$ до $1/2^{32}$.

Обнаружение ошибок при дешифрировании

Иногда получатель хочет проверить, является ли его конкретный ключ правильным ключом симметричного дешифрирования. Если открытый текст сообщения представляет собой что-то похожее на ASCII, он может попытаться расшифровать и прочитать сообщение. Если открытый текст случаен, то существуют другие приемы.

Наивным подходом явилось бы присоединение к открытому тексту до шифрования **проверочного блока** - известного заголовка. Получатель Боб расшифровывает заголовок и проверяет, что он правилен. Это работает, но дает Еве известный кусочек открытого текста, что помогает ей криптоанализировать систему. Это также об-

легчает вскрытие шифров с коротким ключом, таких как DES и все экспортируемые шифры. Рассчитайте заранее один раз для каждого ключа проверочную сумму, затем используйте эту проверочную сумму для определ ения ключа в любом сообщении, которое вы перехватили после этого. *Любая* проверочная сумма ключа, в которую не включены случайные или, по крайней мере, различные данные, обладает этим свойством. По идее это очень похоже на генерацию ключей по ключ евым фразам.

Вот для этого способ получше [821]:

(1) Сгенерите вектор идентификации (отличный от используемого в сообщении).

(2) Используйте этот вектор идентификации для генерации большого блока битов: скажем, 512.

(3) Хэшируйте результат.

(4) Используйте те же фиксированные биты хэш-значения, скажем, 32, для контрольной суммы ключа.

Это тоже дает Еве какую-то информацию, но очень небольшую. Если она попытается использовать младшие 32 бита конечного хэш-значения для вскрытия грубой силой, ей придется для каждого вероятного ключа выпо лнить несколько шифрований и хэширование, вскрытие грубой силой самого ключа окажется быстрее.

Она не получит для проверки никаких известных кусочков открытого текста, и даже если она сумеет подбр осить нам наше же случайное значение, она никогда не получит от нас выбранный открытый текст, так как он будет преобразован хэш-функцией прежде, чем она его увидит.

8.5 Использование ключей

Программное шифрование рискованно. Ушли те дни, когда простые микрокомпьютеры работали под упра влением единственной программы. Сегодня время Macintosh System 7, Windows NT и UNIX. Невозможно сказать, когда операционная система остановит работающую программу шифрования, запишет все на диск и разрешит выполняться какой-то другой задаче. Когда операционная система, наконец, вернется к шифрованию, чтобы там не шифровалось, картинка может оказаться весьма забавной. Операционная система записала пр ограмму шифрования на диск, и ключ записан вместе с ней. Ключ, незашифрованный, будет лежать на диске, пока компьютер не напишет что-нибудь в эту же область памяти поверх. Это может случиться через несколько минут, а может через несколько месяцев. Этого может и никогда не случиться, но ключ все же может оказаться на диске в тот момент, когда жесткий диск густо прочесывается вашим противником. В приоритетной, многозадачной среде, для шифрования можно установить достаточно высокий приоритет, чтобы эта операция не пр ерывалась. Это снизило бы риск. Даже при этом система в целом в лучшем случае ненадежна.

Аппаратные реализации безопаснее. Многие из устройств шифрования разработаны так, чтобы любое вм ешательство приводило бы к уничтожению ключа. Например, в плате шифрования для IBM PS/2 залитый эпо ксидной смолой модуль содержит микросхему DES, батарею и память. Конечно, Вы должны верить, что прои зводитель аппаратуры правильно реализовал все необходимые свойства.

Ряд коммуникационных приложений, например, телефонные шифраторы, могут использовать **сеансовые ключи**. Сеансовым называется ключ, который используется только для одного сеанса связи - единственного телефонного разговора - и затем уничтожается. Нет смысла хранить ключ после того, как он был использован. И если вы используете для передачи ключа от одного абонента другому некоторый протокол обмена ключами, то этот ключ не нужно хранить и перед его использованием. Это значительно снижает вероятность компромет ации ключа.

Контроль использования ключей

В некоторых приложениях может потребоваться контролировать процесс использования сеансового ключа. Некоторым пользователям сеансовые ключи нужны только для шифрования или только для дешифрирования. Сеансовые ключи могут быть разрешены к использованию только на определенной машине или только в опр еделенное время. По одной из схем управления подобными ограничениями к ключу добавляется **вектор контроля** (Control Vector, CV), вектор контроля определяет для этого ключа ограничения его использования (см. раздел 24.1) [1025, 1026]. Этот CV хэшируется, а затем для него и главного ключа выполняется операция XOR. Результат используется как ключ шифрования для шифрования сеансового ключа. Полученный сеансовый ключ затем хранится вместе с CV. Для восстановления сеансового ключа нужно хэшировать CV и выполнить для него и главного ключа операцию XOR. Полученный результат используется для дешифрирования шифрованн ого сеансового ключа.

Преимущества этой схемы в том, что длина CV может быть произвольной, и что CV всегда хранится в открытом виде вместе с шифрованным ключом. Такая схема не выдвигает требований относительно устойчивости аппаратуры к взлому и предполагает отсутствие непосредственного доступа пользователей к ключам. Эта система рассматривается ниже в разделах 24.1 и 24.8.

8.6 Обновление ключей

Представьте себе шифрованный канал передачи данных, для которого вы хотите менять ключи каждый день. Иногда ежедневное распределение новых ключей является нелегкой заботой. Более простое решение - генерировать новый ключ из старого, такая схема иногда называется **обновлением ключа**.

Все, что нужно - это однонаправленная функция. Если Алиса и Боб используют общий ключ и применяют к нему одну и ту же однонаправленную функцию, они получат одинаковый результат. Они могут выбрать из результата нужные им биты и создать новый ключ.

Обновление ключей работает, но помните, что безопасность нового ключа определяется безопасностью старого ключа. Если Еве удастся заполучить старый ключ, она сможет выполнить обновление ключей самостоятельно. Однако, если старого ключа у Евы нет, и она пытается выпо отношению к шифрованному трафику по лнить вскрытие с использованием только шифротекста, обновление ключей является хорошим способом защиты для Алисы и Боба.

8.7 Хранение ключей

Наименее сложными при хранении ключей являются проблемы одного пользователя, Алисы, шифрующей файлы для последующего использования. Так как она является единственным действующим пользователем системы, только она и отвечает за ключ. В некоторых системах используется простой подход: ключ хранится в голове Алисы и больше нигде. Это проблемы Алисы - помнить ключ и вводить его всякий раз, когда ей нужно зашифровать или расшифровать файл.

Примером такой системы является IPS [881]. Пользователи могут либо вводить 64-битовый ключ непосредственно, либо ввести ключ как более длинную символьную строку. В последнем случае система генерирует 64-битовый ключ по строке символов, используя технику перемалывания ключа.

Другим решением является хранить ключ в виде карточки с магнитной полоской, пластикового ключа с встроенной микросхемой ROM (называемого **ROM-ключ**) или интеллектуальной карточки [556, 557, 455]. Пользователь может ввести свой ключ в систему, вставив физический носитель в считывающее устройство, встроенное в его шифрователь или подключенное к компьютерному терминалу. Хотя пользователь может использовать ключ, он не знает его и не может его скомпрометировать. Он может использовать его только тем способом и только для тех целей, которые определены вектором контроля.

ROM-ключ - это очень умная идея. Практически любой способен осознать, что такое физический ключ, каково его значение, и как его защитить. Придание криптографическому ключу некоторой физической формы делает хранение и защиту такого ключа интуитивно более понятным.

Эта техника становится более безопасной при разбиении ключа на две половины, одна из которых хранится в терминале, а вторая - в ROM-ключе. Так работает безопасный телефон STU-III правительства США. Потеря ROM-ключа не компрометирует криптографический ключ - замените этот ключ и все снова станет нормально. То же происходит и при потере терминала. Следовательно, компрометация ROM-ключа или системы не компрометирует криптографический ключ key - врагу нужно заполучить обе части.

Ключи, которые трудно запомнить можно хранить зашифрованными, используя что-то похожее на ключ шифрования ключей. Например, закрытый ключ RSA может быть зашифрован ключом DES и записан на диск. Для восстановления ключа RSA пользователь будет должен ввести ключ DES в программу дешифрирования.

Если ключи генерируются детерминировано (с помощью криптографически безопасного генератора псевдослучайных последовательностей), может быть при помощи легко запоминающегося пароля легче генерировать ключи повторно всякий раз, когда они понадобятся.

В идеале, ключ никогда не должен оказываться вне шифровального устройства в незашифрованном виде. Эта цель не всегда достижима, но к этому нужно стремиться.

8.8 Резервные ключи

Алиса работает главным финансистом в Secrets, Ltd. - "Наш девиз - Мы тебе не скажем." Как примерный служащий корпорации она в соответствии с инструкциями по безопасности шифрует все свои данные. К несчастью, она, проигнорировав инструкции по переходу улицы, попала под грузовик. Что делать президенту компании Бобу?

Если Алиса не оставила копии своего ключа, ему придется несладко. Весь смысл шифрования файлов - в невозможности восстановить их без ключа. Если Алиса не была дурой и не использовала плохих шифровальных программ, то ее файлы пропали навсегда.

У Боба есть несколько способов избежать этого. Простейший иногда называют **условным вручением клю-**

чей (см. раздел 4.14). Он требует, чтобы все сотрудники записали свои ключи на бумажках отдали их начальнику службы безопасности компании, который запрет их где-нибудь в сейф (или зашифрует их главным ключом). Теперь, чтобы не случилось с Алисой, Боб узнает ее ключ у начальника службы безопасности. Еще одну копию Боб также должен хранить в своем сейфе, в противном случае, если начальник службы безопасности попадет под другой грузовик, Бобу снова не повезет.

Проблема такой системы управления ключами в том, что Боб должен верить, что его начальник службы безопасности не воспользуется чужими ключами. Что еще серьезнее, все сотрудники должны верить, что начальник службы безопасности не воспользуется их ключами. Существенно лучшим решением является использование протокола совместного использования секрета (см. раздел 3.7).

Когда Алиса генерирует ключ, она одновременно делит ключ не несколько частей и затем посылает все части - зашифрованные, конечно - различным должностным лицам компании. Ни одна из этих частей сама по себе не является ключом, но все эти части можно собрать вместе и восстановить ключ. Теперь Алиса защищена от злоумышленников, а Боб - от потери всех данных Алисы после ее попадания под грузовик. Или, она может просто хранить разные части, зашифрованные открытыми ключами соответствующих должностных лиц компании, на своем жестком диске. Таким образом, никто не участвует в управлении ключами, пока это не станет необходимым.

Другая схема резервирования [188] использует для временного условного вручения ключей интеллектуальные карточки (см. раздел 24.13). Алиса может поместить ключ, которым закрыт ее жесткий диск, на интеллектуальную карточку и выдать ее Бобу, пока она в отъезде. Боб может использовать карточку для доступа к жесткому диску Алисы, но, так как ключ хранится на карточке, Боб не сможет его узнать. Кроме того, такая система контролируема с обеих сторон: Боб может проверить, что ключ открывает диск Алисы, а когда Алиса вернется, она сможет проверить, использовал ли Боб раз этот ключ, и если да, то сколько раз.

В подобной схеме не нужна передача данных. Для безопасного телефона ключ должен существовать только в течение разговора и не дольше. Для хранилищ данных, как было показано, условное вручение ключей может быть неплохой идеей. Я теряю ключи примерно раз в пять лет, а моя память получше, чем у многих. Если бы 200 миллионов человек пользовались криптографией, подобная частота привела бы к потере 40 миллионов ключей ежегодно. Я храню копии ключей от моего дома у соседа, потому что я могу потерять свои ключи. Если бы ключи от дома были подобны криптографическим ключам, то, потеряв их, я никогда не смог бы попасть внутрь и вступить в свои права владения. Также, как я храню где-то в другом месте копии своих данных, мне имеет смысл хранить и резервные копии моих ключей шифрования.

8.9 Скомпрометированные ключи

Все протоколы, методы и алгоритмы этой книги безопасны только, если ключ (закрытый ключ в системе с открытыми ключами) остается в тайне. Если ключ Алисы украден, потерян, напечатан в газете или скомпрометирован иным способом, то все ее безопасность исчезнет.

Если скомпрометированный ключ использовался для симметричной криптосистемы, Алисе придется изменить свой ключ и надеяться, что случившийся ущерб минимален. Если это закрытый ключ, ее проблемы намного больше, так как ее открытый ключ может храниться на многих серверах в сети. И если Ева получит доступ к закрытому ключу Алисы, она сможет выдать себя за нее в этой сети: читать шифрованную почту, подписывать корреспонденцию и контракты, и так далее. Ева действительно сможет стать Алисой.

Жизненно необходимо, чтобы известие о компрометации закрытого ключа быстро распространилось бы по сети. Нужно немедленно известить все базы данных открытых ключей о случившейся компрометации, чтобы ничего не подозревающий человек не зашифровал сообщение скомпрометированным ключом.

Хорошо, если Алиса знает, когда был скомпрометирован ее ключ. Если ключ распределяет KDC, то Алиса должна сообщить ему о компрометации своего ключа. Если KDC не используется, то ей следует известить всех корреспондентов, которые могут получать от нее сообщения. Кто-то должен опубликовать тот факт, что любое сообщение, полученное после потери ключа Алисой, является подозрительным, и что никто не должен посылать сообщения Алисе, используя соответствующий открытый ключ. Рекомендуется, чтобы программное обеспечение использовало какие-нибудь метки времени, тогда пользователи смогут определить, какие сообщения законны, а какие подозрительны.

Если Алиса не знает точно, когда ее ключ был скомпрометирован, то дело хуже. Алиса может захотеть отказаться от контракта, так как он подписан вместо нее человеком, укравшим у нее ключ. Если система дает такую возможность, то кто угодно сможет отказаться от контракта, утверждая, что его ключ был скомпрометирован перед подписанием. Вопрос должен быть решен арбитром.

Это серьезная проблема показывает, как опасно для Алисы связывать свою личность с единственным ключом. Лучше, чтобы у Алисы были различные ключи для различных приложений - точно также, как она держит в

своем кармане физические ключи для различных замков. Другие решения этой проблемы включают биометрические измерения, ограничения возможностей использования ключа, задержки времени и вторая подпись.

Эти процедуры и рекомендации наверняка не оптимальны, но это лучшее, что мы можем посоветовать. Мораль - защищайте ключи, и сильнее всего защищайте закрытые ключи.

8.10 Время жизни ключей

Ни один ключ шифрования нельзя использовать бесконечно. Время его действия должно истекать автоматически, подробно паспортам и лицензиям. Вот несколько причин этого:

— Чем дольше используется ключ, тем больше вероятность его компрометации. Люди записывают ключи и теряют их. Происходят несчастные случаи. Если вы используете ключ в течение года, то вероятность его компрометации гораздо выше, чем если бы вы использовали его только один день.

— Чем дольше используется ключ, тем больше потери при компрометации ключа. Если ключ используется только для шифрования одного финансового документа на файл-сервере, то потеря ключа означает компрометацию только этого документа. Если тот же самый ключ используется для шифрования всей финансовой информации на файл-сервере, то его потеря гораздо более разрушительна.

— Чем дольше используется ключ, тем больше соблазн приложить необходимые усилия для его вскрытия - даже грубой силой. Вскрытие ключа, используемого в течение дня для связи между двумя воинскими подразделениями, позволит читать сообщения, которыми обмениваются подразделения, и создавать поддельные. Вскрытие ключа, используемого в течение года всей военной командной структурой, позволило бы взломщику в течение года читать все сообщения, циркулирующие в этой системе по всему миру, и подделывать их. В нашем мире закончившейся холодный войны какой ключ выбрали бы для вскрытия вы?

— Обычно намного легче проводить криптоанализ, имея много шифротекстов, шифрованных одним и тем же ключом.

Для любого криптографического приложения необходима стратегия, определяющая допустимое время жизни ключа. У различных ключей могут быть различные времена жизни. Для систем с установлением соединения, таких как телефон, имеет смысл использовать ключ только в течение телефонного разговора, а для нового разговора - использовать новый ключ.

Для систем, использующих специализированные каналы связи, все не так очевидно. У ключей должно быть относительно короткое время жизни, в зависимости от значимости данных и количества данных, зашифрованных в течение заданного периода. Ключ для канала связи со скоростью передачи 1 Гигабит в секунду возможно придется менять гораздо чаще, чем для модемного канала 9600 бит/с. Если существует эффективный метод передачи новых ключей, сеансовые ключи должны меняться хотя бы ежедневно.

Ключи шифрования ключей так часто менять не нужно. Они используются редко (приблизительно раз в день) для обмена ключами. При этом шифротекста для криптоаналитика образуется немного, а у соответствующего открытого текста нет определенной формы. Однако, если ключ шифрования ключей скомпрометирован, потенциальные потери чрезвычайны: вся информация, зашифрованная ключами, зашифрованными ключом шифрования ключей. В некоторых приложениях ключи шифрования ключей заменяются только раз в месяц или даже раз в год. Вам придется как-то уравновесить опасность, связанную с использованием одного и того же ключа, и опасность, связанную с передачей нового ключа.

Ключи шифрования, используемые при шифровании файлов данных для длительного хранения, нельзя менять часто. Файлы могут храниться на диске зашифрованными месяцами или годами, прежде чем они кому-нибудь снова понадобятся. Ежедневное дешифрирование и повторное шифрование новым ключом никак не повысит безопасность, просто криптоаналитик получит больше материала для работы. Решением может послужить шифрование каждого файла уникальным ключом и последующее шифрование ключей файлов ключом шифрования ключей. Ключ шифрования ключей должен быть либо запомнен, либо сохранен в безопасном месте, может быть где-нибудь в сейфе. Конечно же, потеря этого ключа означает потерю всех индивидуальных файловых ключей.

Время жизни закрытых ключей для приложений криптографии с открытыми ключами зависит от приложения. Закрытые ключи для цифровых подписей и идентификации могут использоваться годами (даже в течение человеческой жизни). Закрытые ключи для протоколов бросания монеты могут быть уничтожены сразу же после завершения протокола. Даже если считается, что время безопасности ключа примерно равно человеческой жизни, благоразумнее менять ключ каждую пару лет. Во многих четях закрытые ключи используются только два года, затем пользователь должен получить новый закрытый ключ. Старый ключ, тем не менее, должен храниться в секрете на случай, когда пользователю будет нужно подтвердить подпись, сделанную во время действия старого ключа. Но для подписания новых документов должен использоваться новый ключ. Такая схема по-

зволит уменьшить количество документов, которое криптоаналитик сможет использовать для вскрытия .

8.11 Разрушение ключей

Принимая во внимание, что ключи должны регулярно меняться, старые ключи необходимо уничтожать. Старые ключи имеют определенное значение, даже если они никогда больше не используются. С их помощью враг сможет прочитать старые сообщения, зашифрованные этими ключами [65].

Ключи должны уничтожаться надежно (см. раздел 10.9). Если ключ записан на бумажке, бумажку нужно разрезать и сжечь. Пользуйтесь качественными уничтожителями бумаги, рынок заполнен дефектными устро йствами. Алгоритмы, описанные в этой книге, надежно противостоят вскрытию грубой силой, стоящему милли оны долларов и требующему миллионов лет. Если враг сможет раскрыть ваш ключ, добыв плохо измельченные документы из вашего мусорника и наняв сотню безработных в какой-нибудь отсталой стране за 10 центов в час склеивать вместе кусочки разрезанных страниц, он выгодно вложит пару десятков тысяч долларов .

Если ключ - это микросхема EEPROM, то ключ необходимо переписать несколько раз. Если ключ - это микросхема EPROM или PROM, то она должна быть стерта в порошок и развеяна во все стороны. Если ключ хранится на диске компьютера, действительные биты соответствующего участка памяти должны быть переписаны несколько раз (см. раздел 10.9) или диск должен быть уничтожен.

Возможная проблема состоит в том, что в компьютере ключи могут быть легко скопированы и сохранены во множестве мест. Любой компьютер, реализующий какую-либо схему управления памятью, постоянно выгруж ает программы из памяти и загружает их обратно, усугубляя проблему. Способа гарантировать надежное уни чтожение ключа в компьютере не существует, особенно когда процесс уничтожения контролируется операцио нной системой компьютера. Самым озабоченным необходимо использовать специальную программу, которая на физическом уровне искала бы на диске копию ключа даже в неиспользуемых блоках и затем стирала бы соо тветствующие блоки. Не забывайте также стирать все временных файлов.

8.12 Управление открытыми ключами

Криптография с открытыми ключами упрощает управление ключами, но у нее есть свои собственные пр облемы. У каждого абонента, независимо от числа людей в сети, есть только один открытый ключ . Если Алиса захочет отправить Бобу сообщение, ей придется где-то найти открытый ключ Боба. Она может действовать не сколькими способами:

— Получить ключ от Боба.

— Получить его из централизованной базы данных.

— Получить его из своей личной базы данных.

В разделе 2.5 обсуждаются возможные способы вскрытия криптографии с открытыми ключами , основанных на подмене ключа Боба ключом Мэллори. Используется следующий сценарий: пусть Алиса хочет послать с ообщение Бобу. Она обращается к базе данных открытых ключей и получает открытый ключ Боба . Но подлый Мэллори подменяет ключ Боба своим собственным. (Если Алиса запрашивает ключ непосредственно у Боба, Мэллори для успешной подмены придется перехватить ключ Боба при передаче .) Алиса шифрует сообщение ключом Мэллори и отправляет его Бобу. Мэллори перехватывает сообщение, расшифровывает и читает его . Затем шифрует открытым ключом Боба и отправляет по назначению. Ни Боб, ни Алиса ни о чем не догадыв аются.

Заверенные открытые ключи

Заверенным открытым ключом, или сертификатом, является чей-то открытый ключ, подписанный засл уживающим доверия лицом. Заверенные ключи используются, чтобы помешать попыткам подмены ключа [879]. Заверенный ключ Боба в базе данных открытых ключей состоит не только из открытого ключа Боба . Он содержит информацию о Бобе - его имя, адрес, и т.д. - и подписан кем-то, кому Алиса доверяет - Трентом (обычно известным как **орган сертификации**, certification authority, или CA). Подписав и ключ, и сведения о Бобе, Трент заверяет, что информация о Бобе правильна, и открытый ключ принадлежит ему . Алиса проверяет подпись Трента и затем использует открытый ключ, убедившись в том, что он принадлежит Бобу и никому другому. Заверенные ключи играют важную роль во многих протоколах с открытыми ключами, например, PEM [825] (см. раздел 24.10) и X.509 [304] (см. раздел 24.9).

В таких системах возникает сложная проблема, не имеющая прямого отношения к криптографии . Каков смысл процедуры заверения? Или, иначе говоря, кто для кого имеет полномочия выдавать сертификаты? Кто угодно может заверит своей подписью чей угодно открытый ключ, но должен же быть какой-то способ отфиль тровать ненадежные сертификаты: например, открытые ключи сотрудников компании, заверенные CA другой компании. Обычно создается цепочка передачи доверия: один надежный орган заверяет открытые ключи дов е-

ренных агентов, те сертифицируют CA компании, а CA компании заверяют открытые ключи своих работников. Вот еще вопросы, над которыми стоит подумать:

— Какой уровень доверия к чьей-то личности обеспечивает сертификат?

— Каковы взаимоотношения между человеком и CA, заверяющим его открытый ключ, и как эти отношения отражаются в сертификате?

— Кому можно доверить быть "одним надежным органом", возглавляющим сертификационную цепочку?

— Насколько длинной может быть сертификационная цепочка?

В идеале прежде, чем CA подпишет сертификат Боба, Бобу нужно пройти определенную процедуру авторизации. Кроме того, для защиты от скомпрометированных ключей важно использовать какие-нибудь метки времени или признаки срока действия сертификата [461].

Использование меток времени недостаточно. Ключи могут стать неправильными задолго до истечения их срока либо из-за компрометации, либо по каким-то административным причинам. Следовательно, важно, чтобы CA хранил список неправильных заверенных ключей, а пользователи регулярно сверялись бы с этим списком. Эта проблема отмены ключей все еще трудна для решения.

К тому же, одной пары открытый ключ/закрытый ключ недостаточно. Конечно же, в любая хорошая реализация криптографии с открытыми ключами должна использовать разные ключи для шифрования и для цифровых подписей. Такое разделение разрешает различные Это разделение учитывает различные уровни защиты, сроки действия, процедуры резервирования, и так далее. Кто-то может подписывать сообщения 2048-битовым ключом, который хранится на интеллектуальной карточке и действует двадцать лет, а кто-то может использовать для шифрования 768-битовый ключ, который хранится в компьютере и действует шесть месяцев.

Однако, одной пары для шифрования и одной для подписи также недостаточно. Закрытый ключ может идентифицировать роль человека также, как и личность, а у людей может быть несколько ролей. Алиса может хотеть подписать один документ как лично Алиса, другой - как Алиса, вице-президент Monolith, Inc., а третий - как Алиса, глава своей общины. Некоторые из этих ключей имеют большее значение, чем другие, поэтому они должны быть лучше защищены. Алисе может потребоваться хранить резервную копию своего рабочего ключа у сотрудника отдела безопасности, а она не хочет, чтобы у компании была копия ключа, которым она подписала закладную. Алиса собирается пользоваться несколькими криптографическими ключами точно также, как она использует связку ключей из своего кармана.

Распределенное управление ключами

В некоторых случаях такой способ централизованного управления ключами работать не будет. Возможно, не существует такого CA, которому доверяли бы Алиса и Боб. Возможно, Алиса и Боб доверяют только своим друзьям. Возможно, Алиса и Боб никому не доверяют.

Распределенное управление ключами, используемое в PGP (см. раздел 24.12), решает эту проблему с помощью поручителей. Поручители - это пользователи системы, которые подписывают открытые ключи своих друзей. Например, когда Боб создает свой открытый ключ, он передает копии ключа своим друзьям - Кэрол и Дэйву. Они знают Боба, поэтому каждый из них подписывает ключ Боба и выдает Бобу копию своей подписи. Теперь, когда Боб предъявляет свой ключ чужому человеку, Алисе, он предъявляет его вместе с подписями этих двух поручителей. Если Алиса также знает Кэрол и доверяет ей, у нее появляется причина поверить в правильность ключа Боба. Если Алиса знает Кэрол и Дэйва и хоть немного доверяет им, у нее также появляется причина поверить в правильность ключа Боба. Если она не знает ни Кэрол, ни Дэйва у нее нет причин доверять ключу Боба.

Спустя какое-то время Боб соберет подписи большего числа поручителей. Если Алиса и Боб вращаются в одних кругах, то с большой вероятностью Алиса будет знать одного из поручителей Боба. Для предотвращения подмены Мэллори одного ключа другим поручитель должен быть уверен, прежде чем подписывать ключ, что этот ключ принадлежит именно Бобу. Может быть, поручитель потребует передачи ключа при личной встрече или по телефону.

Выгода этого механизма - в отсутствии CA, которому каждый должен доверять. А отрицательной стороной является отсутствие гарантий того, что Алиса, получившая открытый ключ Боба, знает кого-то из поручителей, и, следовательно, нет гарантий, что она поверит в правильность ключа.

Глава 9
Типы алгоритмов и криптографические режимы

Существует два основных типа симметричных алгоритмов: блочные шифры и потоковые шифры. **Блочные шифры** работают с блоками открытого текста и шифротекста - обычно длиной 64 бита, но иногда длиннее. **Потоковые шифры** работают с битовыми или байтовыми потоками открытого текста и шифротекста (иногда даже с потоками 32-битных слов). Блочный шифр, использующий один и тот же ключ, при шифровании всегда превращает один и тот же блок открытого текста в один и тот же блок шифротекста. Потоковый шифр при каждом шифровании превращает один и тот же бит или байт открытого текста в различные биты или байты шифротекста.

Криптографический **режим** обычно объединяет базовый шифр, какую-то обратную связь и ряд простых операций. Операции просты, потому что безопасность является функцией используемого шифра, а не режима. Более того, режим шифра не должен компрометировать безопасность используемого алгоритма.

Существуют и другие соображения безопасности: должна быть скрыта структура открытого текста, должен быть рандомизирован ввод шифра, должно быть затруднено манипулирование открытым текстом посредством ввода ошибок в шифротекст, должно быть возможно шифрование нескольких сообщений одним ключом. Все это будет подробно рассматриваться в следующих разделах.

Другим важным соображением является эффективность. По эффективности режим не может быть сильно хуже используемого алгоритма. В некоторых обстоятельствах важно, чтобы размер шифротекста совпадал с размером открытого текста.

Третьим соображением является устойчивость к сбоям. Для ряда приложений требуется распараллеливать шифрование или дешифрирование, а другим нужна возможность выполнить как можно большую предобработку. В третьих важно, чтобы процесс дешифрирования умел исправлять сбои битов в потоке шифротекста, а также был устойчив к потере и добавлению битов. Как будет показано, различные режимы обладают различными подмножествами этих характеристик.

9.1 Режим электронной шифровальной книги

Режим **электронной шифровальной книги** (electronic codebook, ECB) - это наиболее очевидный способ использовать блочный шифр: блок открытого текста заменяется блоком шифротекста. Так как один и тот же блок открытого текста заменяется одним и тем же блоком шифротекста, то теоретически возможно создать шифровальную книгу блоков открытого текста и соответствующих шифротекстов. Однако, если размер блока - 64 бита, то кодовая книга будет состоять из 2^{64} записей - слишком много для предварительного вычисления и хранения. И не забывайте, для каждого ключа понадобится отдельная шифровальная книга.

Это самый легкий режим работы. Все блоки открытого текста шифруются независимо. Нет необходимости в последовательном шифровании файла, можно зашифровать сначала 10 блоков из середины текста, затем последние блоки, и наконец, первые. Это важно для шифрованных файлов с произвольным доступом, например, для баз данных. Если база данных зашифрована в режиме ECB, то любая запись может быть добавлена, удалена, зашифрована или расшифрована независимо от любой другой записи (при условии, что каждая запись состоит из целого числа блоков шифрования). Кроме того, обработка может быть распараллелена, если используются несколько шифровальных процессоров, они могут независимо друг от друга шифровать или дешифровать различные блоки.

Проблемой режима ECB является то, что если у криптоаналитика есть открытый текст и шифротекст для нескольких сообщений, он может начать составлять шифровальную книгу, не зная ключа. В большинстве реальных ситуаций фрагменты сообщений имеют тенденцию повторяться. В различных сообщениях могут быть одинаковые битовые последовательности. У сообщений, которые подобно электронной почте создаются компьютером, может быть регулярная структура. Сообщения могут иметь высокую степень избыточности или содержать длинные строки нулей или пробелов.

Если криптоаналитик знает, что блок открытого текста "5e081bc5" при шифровании превращается в блок шифротекста "7ea593a4," то он может мгновенно расшифровать этот блок шифротекста, в каком-бы другом сообщении он не появился. Если в шифрованном сообщении много повторов, которые имеют тенденцию занимать одинаковое место в различных сообщениях, криптоаналитик может получить много информации. Он может попытаться статистически вскрыть используемый открытый текст, независимо от силы блочного шифра.

Особенно уязвимы начало и окончание сообщений, где находится информация об отправителе, получателе дате и т.д. Эта проблема иногда называется **стандартными заголовками** и **стандартными окончаниями**.

Положительной стороной является возможность шифровать несколько сообщений одним ключом без сниже-

ния безопасности. По сути, каждый блок можно рассматривать как отдельное сообщение, шифрованное тем же самым ключом. При дешифрировании битовые ошибки в шифротексте приводят к неправильному дешифрированию соответствующего блока открытого текста, но не влияет на остальной открытый текст . Однако, если бит шифротекста случайно потерян или добавлен, то весь последующий шифротекст будет расшифрован неправильно, если для выравнивания границ блоков не используется какая-нибудь кадровая структура .

Набивка

Большинство сообщений точно не делятся на 64-битные (или любого другого размера) блоки шифрования, в конце обычно оказывается укороченный блок. ECB требует использовать 64-битные блоки. Способом решения этой проблемы является **набивка**.

Последний блок дополняется (набивается) некоторым регулярным шаблоном - нулями, единицами, чередующимися нулями и единицами - для получения полного блока. При необходимости удалить набивку после дешифрирования запишите количество байтов набивки в последний байт последнего блока . Например, пусть размер блока - 64 бита, и последний блок состоит из 3 байтов (24 бит) . Для дополнения блока до 64 бит требуется пять байтов, добавьте четыре байта нулей и последний байт с числом 5. После дешифрирования удалите последние 5 байтов последнего расшифрованного блока. Чтобы этот метод работал правильно, каждое сообщение должно быть дополнено. Даже если открытый текст содержит целое число блоков, вам придется добавить один полный блок. С другой стороны, можно использовать символ конца файла для обозначения последнего байта открытого текста и дополнить этот символ er.

На 8-й показан другой вариант, называемый **похищением шифротекста** [402]. P_{n-1} - последний полный блок открытого текста, а P_n - последний, короткий блок открытого текста. C_{n-1} - последний полный блок шифротекста, и C_n - последний, короткий блок шифротекста. C' - это промежуточный результат, не являющийся частью переданного шифротекста.

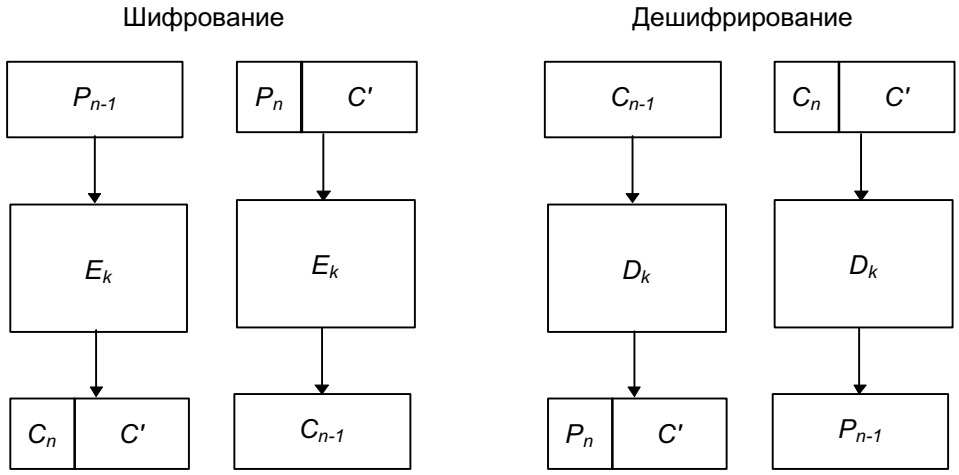


Рис. 9-1. Похищение шифротекста.

9.2 Повтор блока

Более серьезной проблемой режима ECB является то, что враг может изменить шифрованные сообщения, не зная ключа или даже алгоритма, чтобы обмануть предполагаемого получателя. Впервые эта проблемы была рассмотрена в [291].

Для иллюстрации этой проблемы рассмотрим систему передачи денег, которая переводит деньги из банка в банк. Чтобы облегчить жизнь банковских компьютеров, банки согласовали примерно следующий стандартный формат сообщения для передачи денег:

```
Банк 1: Передача            1.5 блока
Банк 2: Прием               1.5 блока
Имя вкладчика               6 блоков
Счет вкладчика              2 блока
Сумма вклада                1 блок
```

Блок соответствует 8-байтному блоку шифрования. Сообщения шифруются с помощью некоторого блочного алгоритма в режиме ECB.

Мэллори, который подслушивает линию связи между банками, банком Алисы и банком Боба, может использовать эту информацию для обогащения. Сначала, он программирует свой компьютер для записи всех шифрованных сообщений из банка Алисы в банк Боба. Затем, он переводит $100 из банка Алисы на свой счет в банк

Боба. Позже, он повторяет эту операцию еще раз. С помощью своего компьютера он проверяет записанные сообщения, разыскивая пару идентичных сообщений. Этими сообщениями являются те сообщения, которыми он переводит $100 на свой счет. Если он находит несколько пар одинаковых сообщений (что больше похоже на реальную жизнь), он делает еще один денежный перевод и записывает результат. В конце концов он сможет выделить сообщение, которым был проведен именно его перевод.

Теперь он может отправить это сообщение по каналу связи, когда захочет. Каждое сообщение приведет к зачислению на его счет в банке Боба еще $100. Когда оба банка сверят свои переводы (возможно в конце дня), они обнаружат переводы-призраки, но если Мэллори достаточно умен, он уже сбежит в какую-нибудь банановую республику без договора об экстрадиции, прихватив с собой деньги. И скорее всего он использует суммы несколько больше $100 и провернет операцию сразу для нескольких банков.

На первый взгляд банки могут легко пресечь это, добавляя метки времени к своим сообщениям.

```
Метка даты/времени          1 блок
Банк 1: Передача            1.5 блока
Банк 2: Прием               1.5 блока
Имя вкладчика               6 блоков
Счет вкладчика              2 блока
Сумма вклада                1 блок
```

В такой системе два идентичных сообщения будут легко обнаружены. Тем не менее, с помощью метода, называемого **повтором блока**, Мэллори все же сможет обогатиться. На 7-й показано, что Мэллори может собрать восемь блоков шифротекста, соответствующих его имени и номеру счета: блоки с 5 по 12. В этот момент уместно дьявольски рассмеяться, ведь Мэллори уже в полной готовности.

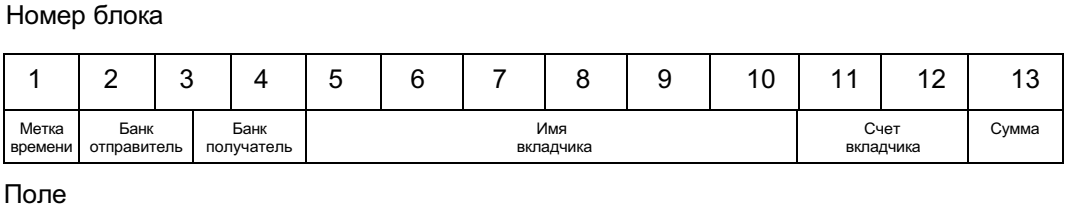


Рис. 9-2. Блоки шифрования в записи приведенного примера.

Он перехватывает сообщения из банка Алисы в банк Боба и заменяет блоки с 5 по 12 сообщения байтами, соответствующими его имени и номеру счета. Затем он посылает измененные сообщения в банк Боба. Ему не нужно знать, кто был отправителем денег, ему даже не нужно знать переводимую сумму (хотя он может связать подправленное сообщение с соответствующим увеличением своего счета и определить блоки, соответствующие определенным денежным суммам). Он просто изменяет имя и номер счета на свои собственные и следит за ростом своих доходов. (Я помню, что Мэллори надо быть осторожным, чтобы не модифицировать сообщение о снятии денег, но предположим на минутку, что у этих сообщений другая длина или иной отличительный признак.)

Для обнаружения такого способа банкам одного дня не хватит. Когда они сверят свои переводы в конце дня, все суммы совпадут. Возможно, пока настоящий вкладчик не заметит, что его вклады не зачисляются на счет, или пока кто-нибудь не обратит внимание на неожиданную активизацию работы со счетом Мэллори, банки не смогут заметить никаких следов. Мэллори не глуп и к этому времени закроет свой счет, изменит имя и купит виллу в Аргентине.

Банки могут минимизировать эту проблему, часто меняя свои ключи, но это означает только, что Мэллори придется действовать побыстрее. Однако, добавление MAC также решит проблему. Несмотря на это рассматриваемая проблема фундаментальна для режима ECB. Мэллори удалять, повторять или заменять блоки по своему усмотрению. Решением является способ, называемый **сцеплением**.

9.3 Режим сцепления блоков шифра

Сцепление добавляет к блочному шифру механизм обратной связи: результаты шифрования предыдущих блоков влияют на шифрование текущего блока. Другими словами, каждый блок используется для изменения шифрования следующего блока. Каждый блок шифротекста зависит не только от шифруемого блока открытого текста, но и от всех предыдущих блоков открытого текста.

В режиме **сцепления блоков шифра** (cipher block chaining, CBC) перед шифрованием над открытым текстом и предыдущим блоком шифротекста выполняется операция XOR. На 6-й (а) показано шифрование CBC в действии. ,Когда блок открытого текста зашифрован, полученный шифротекст сохраняется в регистре обратной связи. Прежде чем будет зашифрован следующий блок открытого текста, он подвергается операции XOR вместе

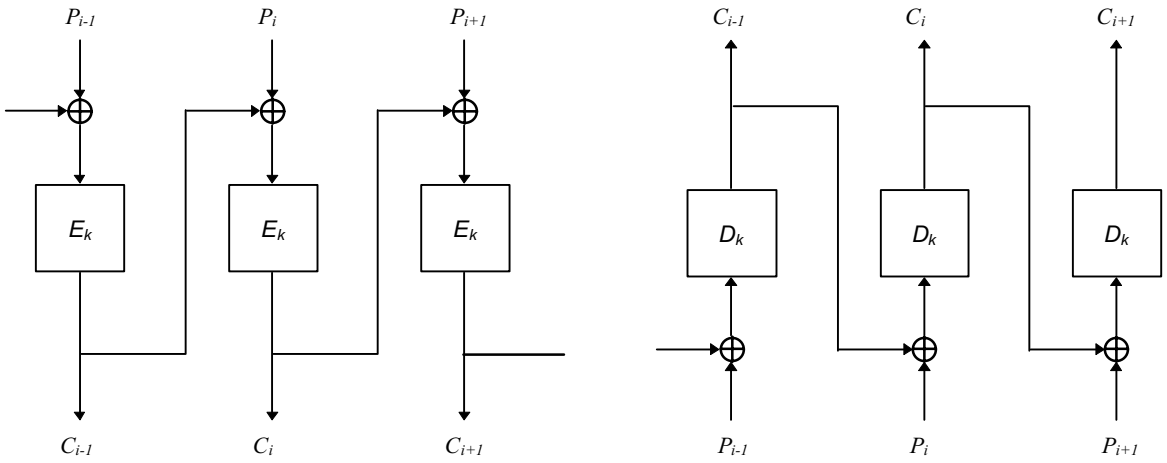
с содержимым регистра обратной связи. Таким образом создаются входные данные для следующего этапа процедуры шифрования. Полученный шифротекст снова сохраняется в регистре обратной связи, чтобы подвергнуться операции XOR вместе со следующим блоком открытого текста, и так до конца сообщения. Шифрование каждого блока зависит от всех предыдущих блоков.

Дешифрирование является обратной операцией (см. Figure 9.3 (б)). Блок шифротекста расшифровывается как обычно, но сохраняется в регистре обратной связи. Затем следующий блок дешифрируется и подвергается операции XOR вместе с содержимым регистра обратной связи. Теперь следующий блок шифротекста сохраняется в регистре обратной связи, и так далее, до конца сообщения.

Математически это выглядит следующим образом:

$$C_i = E_K(P_i \oplus C_{i-1})$$

$$P_i = C_{i-1} \oplus D_K(C_i)$$



(а) Шифрование CBC

(б) Дешифрирование CBC

Рис. 9-3. Режим сцепления блоков шифра.

Вектор инициализации

В режиме CBC одинаковые блоки открытого текста при шифровании переходят в различные блоки шифротекста только, если отличались какие-то из предшествующих блоков открытого текста. Два идентичных сообщения, однако, будут шифроваться как один и тот же шифротекст. Что еще хуже, два одинаково начинающихся сообщения будут шифроваться одинаково, пока не появится первое различие.

У ряда сообщений может быть одинаковый заголовок - тема письма, строка "From" или еще что-нибудь. Хотя повтор блока будет невозможен, такое одинаковое начало может предоставить криптоаналитику какую-нибудь полезную информацию.

Избежать этого можно, шифруя в качестве первого блока какие-то случайные данные. Этот блок случайных данных называется вектором инициализации (initialization vector, IV), инициализирующей переменной или начальным значением сцепления. IV не имеет никакого смыслового значения, он используется только для того, чтобы сделать каждое сообщение уникальным. Когда получатель расшифровывает этот блок, он испольтзует его только для заполнения регистра обратной связи. Хорошим IV служит метка времени. Или используйте какие-нибудь случайные биты.

С использованием IV сообщения с идентичным открытым текстом при шифровании переходят в сообщения с различным шифротекстом. Следовательно, злоумышленник не сможет предпринять повтор блока, и затруднится создание шифровальной книги. Хотя рекомендуется для каждого сообщения, шифруемого одним и тем же ключом, выбирать уникальный IV, это требование не является обязательным.

IV не должен храниться в секрете, он может передаваться открыто вместе с шифротекстом. Если вы не понимаете почему, рассмотрите следующий довод. Пусть наше сообщение состоит из нескольких блоков: B_1, B_2, ..., B_i. B_1 шифруется вместе с IV. B_2 шифруется с использованием шифротекста B_1 в роли IV. B_3 шифруется с использованием шифротекста B_2 в роли IV, и так далее. Итак, если количество блоков - n, то $n-1$ "векторов инициализации" открыты, даже если первоначальный IV хранится в секрете. Поэтому причин хранить в секрете IV нет, IV - это просто блок-заглушка, можно считать его нулевым блоком сцепления B_0.

Набивка

Набивка используется также, как и в режиме ECB, но в некоторых приложениях размер шифротекст должен в точности совпадать с размером открытого текста. Может быть, зашифрованный файл должен занять в точности тот же объем памяти, что и файл открытого текста. В этом случае последний короткий блок придется шифровать иначе. Пусть последний блок состоит из *l* битов. Зашифровав последний полный блок, снова зашифруйте шифротекст, выберите старшие *l* битов и выполните для них и короткого блока операцию XOR, создавая шифротекст. Эта процедура показана на 5-й.

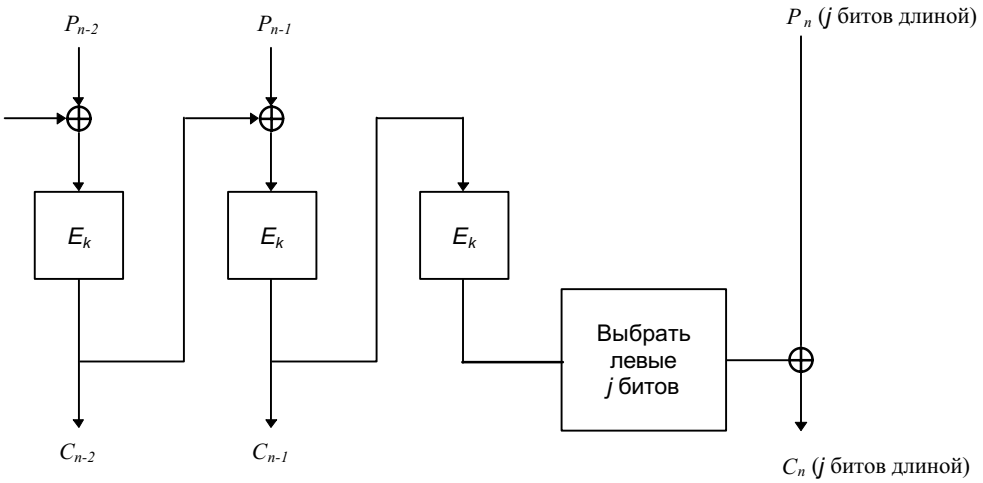


Рис. 9-4. Шифрование короткого последнего блока в режиме CBC.

Слабость этого способа в том, что хотя Мэллори не сможет раскрыть последний блок шифротекста, он может систематически изменять его, меняя отдельные биты шифротекста. Если последние несколько битов шифротекста содержат важную информацию, это опасно. Если последние биты просто содержат совет по домоводству, то ничего страшного.

Лучшим способом является похищение щифротекста (см. 4th) [402]. P_{n-1} - последний полный блок открытого текста, P_n - заключительный, короткий блок открытого текста. C_{n-1} - последний полный блок шифротекста, C_n - заключительный, короткий блок шифротекста. C' - это просто промежуточный результат, не являющийся частью переданного шифротекста. Преимуществом этого метода является то, что все биты открытого текста сообщения проходят через алгоритм шифрования.

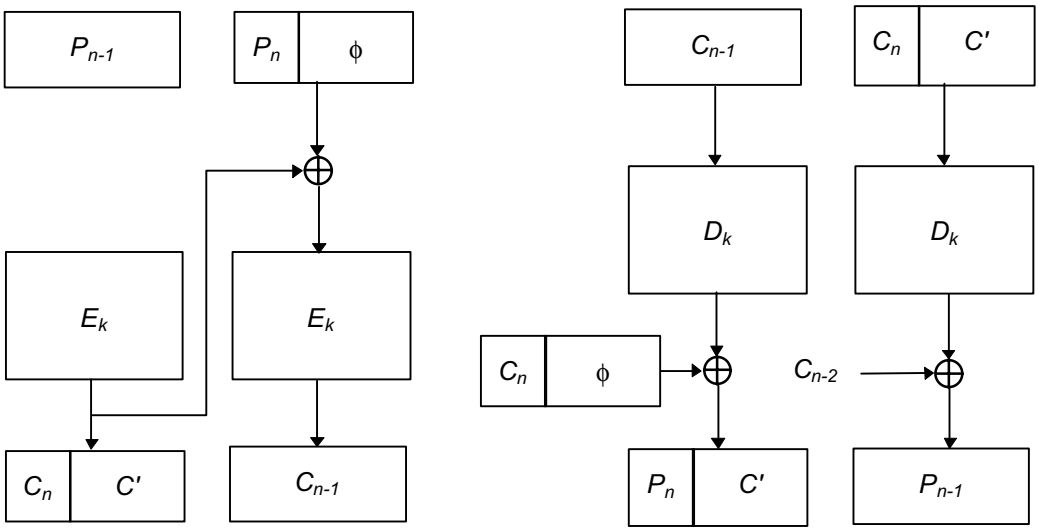


Рис. 9-5. Похищение шифротекста в режиме CBC.

Распространение ошибки

Режим CBC характеризуется **прямой обратной связью** шифротекста при шифровании и **инверсной обратной связью** шифротекста при дешифрировании. При этом приложения должны уметь бороться с ошибками. Единственная битовая ошибка в блоке открытого текста повлияет на данный блок шифротекста и все последующие блоки шифротекста. Это не важно, потому что дешифрирование инвертирует этот эффект, и восстано-

ленный открытый текст будет содержать ту же единственную ошибку.

Чаще встречаются ошибки шифротекста. Они легко появляются из-за шума линий передачи или сбоев устройств хранения. В режиме CBC ошибка одного бита шифротекста влияет на один блок и один бит восстановленного открытого текста. Блок, соответствующий содержащему ошибку блоку шифротекста, искажается полностью. В следующем блоке искажается единственный бит, находящийся в той же позиции, что и ощибочный бит.

Это свойство превращения малой ошибки шифротекста в большую ошибку открытого текста называется **распространением ошибки**. Это является главным недостатком. Эта ошибка не влияет на блоки, расположенные через один от испорченного и далее, поэтому режим CBC является **самовосстанавливающимся**. Ошибка влияет на два блока, но система продолжает работать правильно для всех последующих блоков. CBC представляет собой пример блочного шифра, используемого в самосинхронизирующейся манере, но только на блоковом уровне.

Хотя режим CBC быстро восстанавливается от битового сбоя, он абсолютно не устойчив к ошибкам синхронизации. Если в потоке шифротекста теряется или добавляется бит, то положение всех последующих блоков сдвигаются на один бит, и на выходе дешифрирования будет сплошной мусор. Любая криптосистема, использующая режим CBC должна обеспечивать целостность блочной структуры либо при помощи кадров, либо сохраняя данные в структуры из нескольких блоков.

Вопросы безопасности

Ряд возможных проблем обуславливаются структурой CBC. Во первых, так как блок шифротекста достаточно просто влияет на следующий блок, Мэллори может тайно добавлять блоки к концу зашифрованного сообщения. Конечно, при дешифрировании они превратятся в чепуху, но в некоторых ситуациях это нежелательно.

При использовании CBC вы должны структурировать ваш открытый текст так, чтобы вы знали, где находятся концы сообщений, и могли обнаружить добавление лишних блоков.

Во вторых, Мэллори может изменить блок шифротекста, изменения определенным образом блоки расшифрованного открытого текста. Например, если Мэллори изменит один бит шифротекста, весь блок будет расшифрован неправильно, а в следующем блоке в соответствующей позиции будет неправильный бит. Возможны ситуации, когда это нежелательно. Открытое сообщения должно обладать некоторой избыточностью или средствами идентификации.

Наконец, хотя структура открытого текста маскируется сцеплением, структура очень длинных сообщений все равно будет заметна. Парадокс дня рождения предсказывает, что после $2^{m/2}$ блоков, где m - размер блока, появляются одинаковые блоки. Для 64-битового блока длина такого сообщения примерно равны 32 Гбайтам. Подобная проблема возникает только для сообщений немаленького размера.

9.4 Потоковые шифры

Потоковые шифры преобразуют открытый текст в шифротекст по одному биту за операцию. Простейшая реализация потокового шифра показана на 3-й. **Генератор потока ключей** (иногда называемый генератором с бегущим ключом) выдает поток битов: k_1, k_2, k_3, ..., k_i. Этот поток ключей (иногда называемый бегущим ключом) и поток битов открытого текста, p_1, p_2, p_3, ..., p_i, подвергаются операции "исключающее или", и в результате получаетсяы поток битов шифротекста.

$c_i = p_i \oplus k_i$

При дешифрировании операция XOR выполняется над битами шифротекста и тем же самым потоком ключей для восстановления битов открытого текста.

$p_i = c_i \oplus k_i$

Так как

$p_i \oplus k_i \oplus k_i = p_i$

это работает правильно.

Безопасность системы полностью зависит от свойств генератора потока ключей. Если генератор потока ключей выдает бесконечную строку нулей, шифротекст будет совпадать с открытым текстом, и все операция будет бессмысленна. Если генератор потока ключей выплевывает повторяющийся 16-битовый шаблон, алгоритм будет являться простым XOR с пренебрежимо малой безопасностью (см. раздел 1.4). Если генератор потока ключей выплевывает бесконечный поток случайных (по настоящему, а не псевдослучайных - см. раздел 2.8) битов, вы получаете одноразовый блокнот и идеальную безопасность.

На деле безопасность потокового шифра находится где-то между простым XOR и одноразовым блокнотом.

Генератор потока ключей создает битовый поток, который похож на случайный, но в действительности детерминирован и может быть безошибочно воспроизведен при дешифрировании. Чем ближе выход генератора потока ключей к случайному, тем больше времени потребуется криптоаналитику, чтобы взломать шифр.

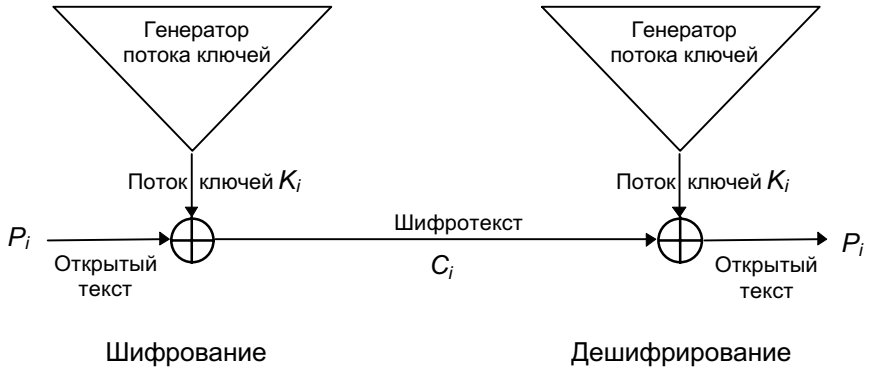


Рис. 9-6. Потоковый шифр

Однако, если генератор потока ключей при каждом включении создает один и тот же битовый поток, то использующую его криптосистему взломать нетрудно. Покажем на примере, почему это так.

Если к Еве попал шифротекст и соответствующий открытый текст, то она, выполняя операцию XOR над открытым текстом и шифротекстом, раскрывает поток ключей. Или, если у нее есть два различных шифротекста, зашифрованных одинаковым ключом, она может выполнить над ними операцию XOR, получая два открытых текста сообщений, над которыми выполнена операция XOR. Это нетрудно взломать, и затем она может получить поток ключей, выполняя операцию XOR над одним из открытых текстов и шифротекстом.

Теперь, перехватив любое другое шифрованное сообщение, она сможет расшифровать его, используя полученный поток ключей. Кроме того, она может расшифровать и прочитать любое из ранее перехваченных сообщений. Когда Ева получит пару открытый текст/шифротекст, она сможет читать все.

Поэтому для всех потоковых шифров используются ключи. Выход генератора потока ключей является функцией ключа. Теперь, если Ева получит пару открытый текст/шифротекст, она сможет читать только те сообщения, которые зашифрованы тем же ключом. Измените ключ, и противнику придется начать все сначала. Потоковые шифры особенно полезны для шифрования бесконечных потоков коммуникационного трафика, например, канала T1, связывающего два компьютера.

Генератор потока ключей состоит из трех основных частей (см. 2nd). Внутреннее состояние описывает текущее состояние генератора потока ключей. Два генератора потока ключей, с одинаковым ключом и одинаковым внутренним состоянием, выдают одинаковые потоки ключей. Функция выхода по внутреннему состоянию генерирует бит потока ключей. Функция следующего состояния по внутреннему состоянию генерирует новое внутреннее состояние.

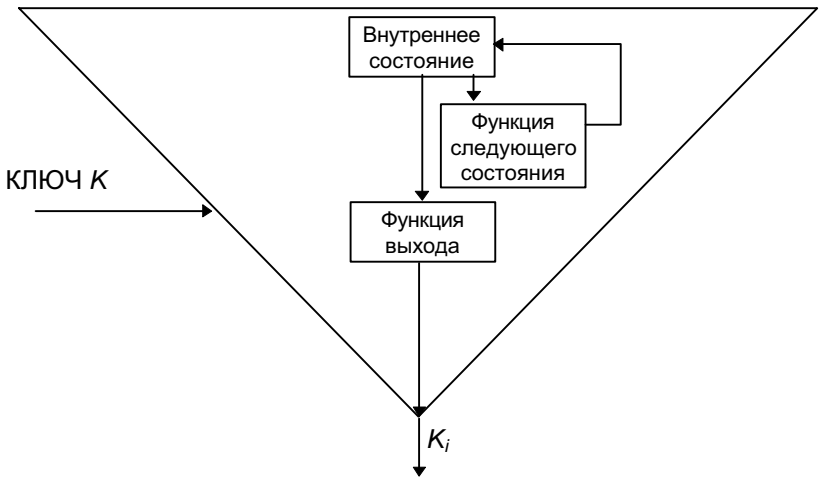


Рис. 9-7. Устройство генератора потока ключей.

9.5 Самосинхронизирующиеся потоковые шифры

В **самосинхронизирующихся потоковых шифрах** каждый бит потока ключей является функцией фиксированного числа предыдущих битов шифротекста [1378]. Военные называют этот шифр **автоключом шифротекста** (ciphertext auto key, CTAK). Основная идея была запатентована в 1946 [667].

Самосинхронизирующийся потоковый шифр показан на 1-й. Внутреннее состояние является функцией предыдущих n битов шифротекста. Криптографически сложной является выходная функция, которая использует внутреннее состояние для генерации бита потока ключей.

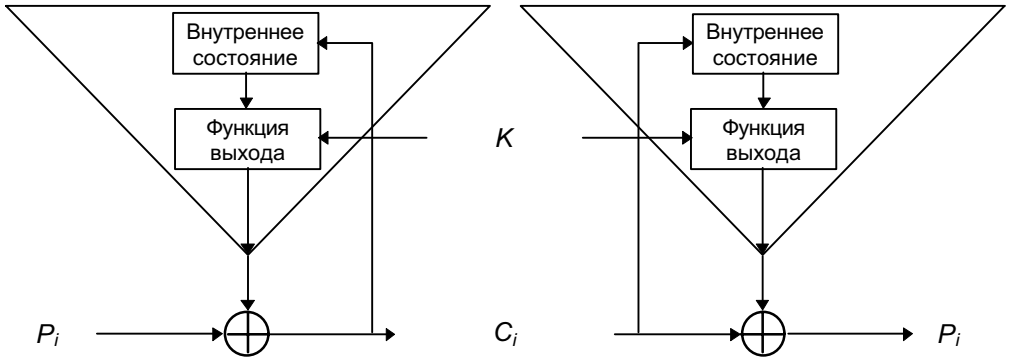


Рис. 9-8. Самосинхронизирующийся генератор потока ключей.

Так как внутреннее состояние полностью зависит от предыдущих n шифротекста, дешифрирующий генератор потока ключей автоматически синхронизируется с шифрующим генератором потока ключей, приняв n битов шифротекста.

В интеллектуальных реализациях этого режима каждое сообщение начинается случайным заголовком длиной n битов. Этот заголовок шифруется, передается и затем расшифровывается. Расшифровка будет неправильной, но после этих n битов оба генератора потока ключей будут синхронизированы.

Слабой стороной самосинхронизирующегося потокового шифра является распространение ошибки. Для каждого бита шифротекста, испорченного при передаче, дешифрирующий генератор потока ключей выдает n неправильных битов потока ключей. Следовательно, каждому неправильному биту шифротекста соответствуют n ошибок в открытом тексте, пока испорченный бит не перестанет влиять на внутреннее состояние.

Вопросы безопасности

Самосинхронизирующиеся потоковые шифры также чувствительны к вскрытию повторной передачей. Сначала Мэллори записывает несколько битов шифротекста. Затем, позднее, он вставляет эту запись в текущий трафик. После выдачи некоторой чепухи, пока принимающая сторона синхронизируется с вставленной записью, старый шифротекст будет расшифрован как нормальный. У принимающей стороны нет способа узнать, что полученные данные являются повторно передаваемой записью. Если не используются метки времени, Мэллори может убедить банк снова и снова зачислять деньги на его счет, повторно передавая одно и то же сообщение (конечно, при условии, что ключ не менялся). Другие слабые места этой схемы могут стать заметны при очень частой пересинхронизации [408].

9.6 Режим обратной связи по шифру

Блочный шифр также может быть реализованы как самосинхронизирующийся потоковый шифр, такой режим называется режимом обратной связи по шифру (cipher-feedback, CFB). В режиме CBC шифрование не могло начаться, пока не получен целый блок данных. Это создает проблемы для некоторых сетевых приложений. Например, в безопасной сетевой среде терминал должен иметь возможность передавать главному компьютеру каждый символ сразу, как только он введен. Если данные нужно обрабатывать байтами, режим CBC также не работает.

В режиме CFB единица зашифрованных данных может быть меньше размера блока. В следующем примере каждый раз шифруется только один символ ASCII (это называется 8-битовым шифрованием), но в числе 8 нет ничего волшебного. Вы можете шифровать данные по одному биту с помощью 1-битового CFB, хотя использование для единственного бита полного шифрования блочным шифром потребует много ресурсов, потоковый шифр в этом случае был бы идеей получше. (Уменьшение количества циклов блочного фильтра для повышения скорости не рекомендуется [1269].) Можно также использовать 64-битовый CFB, или любой n-битовый CFB, где n больше или равно размеру блока.

На 0-й показан 8-битовый режим CFB, работающий с 64-битовым алгоритмом. Блочный алгоритм в режиме CFB работает с очередью, размер которой равен размеру используемого блока. Сначала очередь заполнена IV, как и в режиме CBC. Очередь шифруется и для крайних левых восьми битов результата выполняется XOR с первыми 8-битовым символом открытого текста для получения первого 8-битового символа шифротекста. Теперь этот символ передается. Те же восемь битов также передвигаются на место крайних правых восьми битов очереди, а крайними левыми битами становятся следующие восемь битов. Крайние восемь левых битов отбрасывается. Следующий символ открытого текста шифруется тем же способом. Дешифрирование является обратным процессом. И шифрующей, и дешифрирующей стороной блочный алгоритм используется в режиме шифрования.

Если размер блока алгоритма - n, то -битовый CFB выглядит следующим образом (см. -1-й):

$C_i = P_i \oplus E_k(C_{i-1})$

$P_i = C_i \oplus E_k(C_{i-1})$

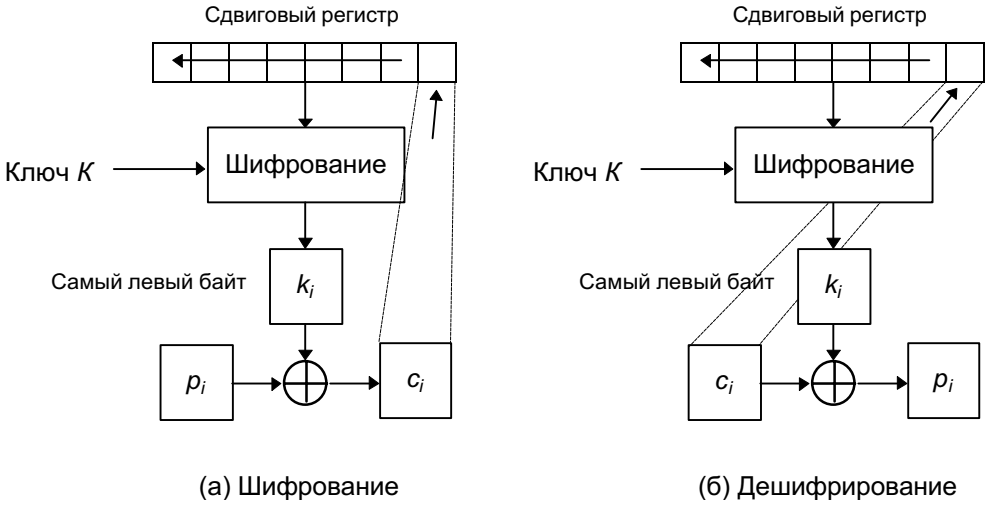


Рис. 9-9. Режим 8-битовой обратной связи по шифру.

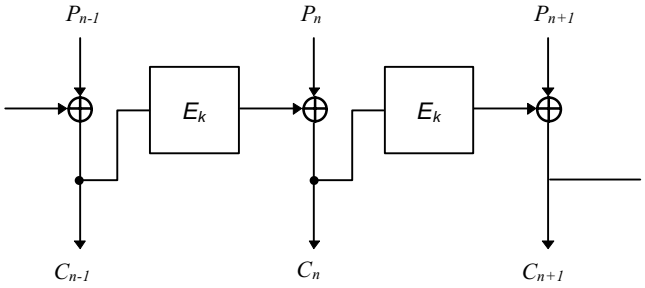


Рис. 9-10. n-битовый CBF с n-битовым алгоритмом.

Как и режим CBC, режим CFB связывает вместе символы открытого текста так, что шифротекст зависит от всего предшествующего открытого текста.

Вектор инициализации

Для инициализации процесса CFB в качестве входного блока алгоритма может использоваться вектор инициализации IV. Как и в режиме CBC IV не нужно хранить в секрете.

Однако IV должен быть уникальным. (В отличие от режима CBC, где IV не обязан быть уникальным, хотя это и желательно.) Если IV в режиме CFB не уникален, криптоаналитик может раскрыть соответствующий открытый текст. IV должен меняться для каждого сообщения. Это может быть последовательный номер, увеличивающийся для каждого нового сообщения и не повторяющийся в течение времени жизни ключа. Если данные шифруются с целью последующего хранения, IV может быть функцией индекса, используемого для поиска данных.

Распространение ошибки

В режиме CFB ошибка в открытом тексте влияет на весь последующий шифротекст, но самоустраняется при дешифрировании. Гораздо интереснее ошибка в шифротексте. Первым эффектом сбоя бита шифротекста является сбой одного бита открытого текста. Затем ошибка попадает в сдвиговый регистр, и пока сбойный бит не выйдет из регистра, будет формироваться неправильный шифротекст. В 8-битовом режиме CFB из-за сбоя единственного бита портятся 9 байтов расшифрованного открытого текста. Потом система восстанавливается, и весь последующий шифротекст расшифровывается правильно. В общем случае в *n*-битовом режиме CFB одна ошибка шифротекста влияет на дешифрирование текущего и следующих *m/n*-l блоков, где *m* - размер блока.

Более тонкой проблемой, связанной с такого рода распространением ошибки, является то, что если Мэллори знает открытый текст сообщения, он может поиграть битами данного блока, заставляя их расшифровываться в нужные ему данные. *Следующий* блок при дешифрировании превратится в чепуху, но вред уже будет причинен. К тому же, он может, оставаясь необнаруженным, менять последние биты сообщения.

CFB самовосстанавливается и после ошибок синхронизации. Ошибка попадает в сдвиговый регистр и, пока она находится там, портит 8 байтов данных. CFB представляет собой пример блочного шифра, который можно использовать как самосинхронизирующийся потоковый шифр (на уровне блоков).

9.7 Синхронные потоковые шифры

В **синхронном потоковом шифре** поток ключей генерируется независимо от потока сообщения. Военные называют этот шифр **ключевым автоключом** (Key Auto-Key, KAK). При шифровании генератор потока ключей один за другим выдает биты потока ключей. При дешифрировании другой генератор потока ключей один за другим выдает идентичные биты потока ключей. Это работает, если оба генератора синхронизированы. Если один из них пропускает один из циклов, или если бит шифротекста теряется при передаче, то после ошибки каждый символ шифротекста будет расшифрован неправильно.

Если такое случается, отправитель и получатель должны повторно синхронизировать свои генераторы потока ключей прежде, чем можно будет продолжить работу. Что еще хуже, они должны выполнить синхронизацию так, чтобы ни одна часть потока ключей не была повторена, поэтому очевидное решение перевести генератор в более раннее состояние не работает.

Положительная сторона синхронных фильтров - это отсутствие распространения ошибок. Если при передаче бит изменит свое значение, что намного вероятнее его потери, то только испорченный бит будет дешифрован неправильно. Все предшествующие и последующие биты не изменятся.

Генератор должен выдавать один и тот же поток ключей и для шифрования, и для дешифрирования, следовательно, выход генератора должен быть предопределен. Если он реализуется на конечном автомате (т.е., компьютере), последовательность со временем повторится. Такие генераторы потока ключей называются **периодическими**. За исключением одноразовых блокнотов все генераторы потока ключей являются периодическими.

Генератор потока ключей должен обладать длинным периодом, намного более длинным, чем количество битов, выдаваемых между сменой ключей. Если период меньше, чем размер открытого текста, то различные части открытого текста будут зашифрованы одинаковым образом, что сильно ослабляет безопасность системы. Если криптоаналитику известна часть открытого текста, он может раскрыть часть потока ключей и использовать ее для дальнейшего раскрытия открытого текста. Даже если у аналитика есть только шифротекст, он может выполнить XOR над разделами, шифрованными одинаковым потоком ключей, и получить XOR соответствующих участков открытого текста. При этом используемый алгоритм превращается в простой алгоритм XOR с очень длинным ключом.

Конкретная длина периода зависит от приложения. Генератор потока ключей, шифрующий непрерывный канал T1, будет шифровать $2^?$ бит в день. Период генератора должен быть на несколько порядков больше этого значения, даже если ключ меняется ежедневно. Если период имеет достаточную длину, ключ можно будет менять раз в неделю или даже раз в месяц.

Синхронные потоковые шифры также предохраняют от любых вставок и удалений шифротекста, так как они приводят к потере синхронизации и будут немедленно обнаружены. Однако, они не защищают полностью от битовых сбоев. Как и при блоковых шифрах в режиме CFB, Мэллори может изменить отдельные биты потока. Если ему известен открытый текст, он может изменить эти биты так, чтобы эти биты дешифрировались так, как ему надо. Дальнейшие биты при дешифрировании превратятся в чепуху (пока система не восстановится), но в определенных приложениях Мэллори может принести заметный ущерб.

Вскрытие вставкой

Синхронные потоковые шифры чувствительны к **вскрытию вставкой** [93]. Пусть Мэллори записал поток шифротекста, но не знает ни открытого текста, ни потока ключей, использованного для шифрования открытого

текста.

Оригинальный открытый текст: pl *p!* p3 *Pi* Оригинальный поток ключей: *kl k! kj ki* Оригинальный шифротекст: *cl c!* c3 *ci*

Мэллори вставляет один известный ему бит, w', в открытый текст после pl и затем пытается получить модифицированный открытый текст, шифрованный тем же потоком ключей. Он записывает получившийся новый шифротекст:

Новый открытый текст: pl *p' pl pi pi* Оригинальный поток: *k. k! k-i ks k!,* Обновленный шифротекст: cl c'z c'3 *c'i* c'i

Так как он знает значение *p'*, он может определить весь открытый текст после этого бита по оригинальному и новому шифротекстам:

k! = *c'z s p',* затем *p!* = *c! s k! kj* = c'3 *S pt,* затем p3 = c3 *S* fc3 *kt* = c', *S* p3, затем p,, = *cs S ks*

Мэллори даже не нужно знать точное положение вставленного бита, он может просто сравнить оригинальный и обновленный шифротексты, чтобы обнаружить, где они начинают отличаться. Для предотвращения такого вскрытия никогда не используйте один поток ключей для шифрования двух различных сообщений.

9.8 Режим выходной обратной связи

Режим выходной обратной связи (Output-feedback, OFB) представляет собой метод использования блочного шифра в качестве синхронного потокового шифра. Этот режим похож на CFB за исключением того, что *n* битов предыдущего выходного блока сдвигаются в крайние правые позиции очереди (см. -2nd). Дешифрирование является обратным процессом. Такой режим называется *n*-битовым OFB. И при шифровании, и при дешифровании блочный алгоритм работает в режиме шифрования. Это иногда называют **внутренней обратной связью**, потому что механизм обратной связи не зависит ни от потоков открытого текста, ни от потоков шифротекста [291]. Если размер блока алгоритма *n*, то *n*-битовый алгоритм OFB выглядит, как показано на :

C, = P, © *S,! S,* = *I, - I,) *P,* = C, © *Sh Si* = *Ek*Si,*

I,)

s - состояние, независящее ни от открытого текста, ни от шифротекста. К числу положительных свойств OFB относится то, что большая часть работы может быть выполнена автономно, даже до того, как появится открытый текст сообщения. Когда наконец сообщение наконец появится, для получения шифротекста над сообщением и выходом алгоритма нужно будет выполнить операцию XOR.

Рис. 9-11. 8-битовый режим

Вектор инициализации

В сдвиговый регистр OFB также сначала должен быть загружен IV. Он должен быть уникальным, но сохранять его в секрете не обязательно.

Распространение ошибки

В режиме OFB распространения ошибки не происходит. Неправильный бит шифротекста приводит к неправильному биту открытого текста. Это может быть полезно при цифровой передаче аналоговых величин, например оцифрованного звука или видеоизображения, когда случайный сбой бита допустим, но распространение ошибки нежелательно.

С другой стороны, потеря синхронизации смертельна. Если сдвиговые регистры при шифровании и при дешифрировании отличаются, то восстановленный открытый текст представляет собой бессмыслицу. Любая система, использующая режим OFB, должна включать механизм обнаружения потери синхронизации и механизм заполнения обоих сдвиговых регистров новым (или одинаковым) IV для восстановления синхронизации.

Рис. 9-12. n-битовый OFB с n-битовым алгоритмом.

OFB и проблемы безопасности

Анализ режима OFB [588, 430, 431, 789] показывает, что OFB стоит использовать только, когда размер обратной связи совпадает с размером блока. Например, 64-битовый алгоритм нужно использовать только в 64-битовом режиме OFB. Несмотря на то, что правительство США разрешает для DES и другие размеры обратных

связей DES [1143], избегайте их.

Режим OFB выполняет XOR над потоком ключей и текстом. Этот поток ключей со временем повторяется. Важно, чтобы он не повторялся для того же ключа, в противном случае нарушается безопасность . Когда размер обратной связи равен размеру блока, блочный шифр переставляет *m*-битовые значения (где *m* - это размер блока), и средняя длина цикла составляет 2™ -1. При длине блока 64 бита это очень большое число. Когда размер обратной связи *n* меньше длины блока, средняя длина цикла падает до приблизительно 2'"*. Для 64-битного шифра это только * - что явно недостаточно.

Потоковые шифры в режиме OFB

Потоковые шифры также могут работать в режиме OFB. В этом случае ключ влияет на функцию следующего состояния (см. -4-й). Функция выхода не зависит от ключа, очень часто она является чем-то простым, напр имер, одним битом внутреннего состояния или результатом XOR нескольких битов внутреннего состояния. Криптографически сложной является функция следующего состояния, которая зависит от ключа . Этот метод также называется внутренней обратной связью [291], потому что механизм обратной связи является вложенным по отношению к алгоритму генерации ключей.

Рис. 9-13. Генератор потока ключей в режиме с выходной обратной связью.

В одном из вариантов этого режима ключ определяет только начальное состояние генератора потока ключей . После того, как ключ определит внутреннее состояние генератора, генератор работает, не подвергаясь воздейс твиям извне.

9.9 Режим счетчика

Блочные шифры в **режиме счетчика** используют в качестве входов алгоритма последовательные номера [824, 498, 715]. Для заполнения регистра используется счетчик, а не выход алгоритма шифрования . После шифрования каждого блока счетчик инкрементируется на определенную константу, обычно единицу . Для этого режима свойства синхронизации и распространения ошибки такие же, как и для OFB. Режим счетчика решает проблему *n*-битового выхода режима OFB, где *n* меньше длины блока.

К счетчику не предъявляется никаких особых требований, он не должен проходить по порядку все возмо жные значения. В качестве входа блочного алгоритма можно использовать генераторы случайных чисел, описа нные в главах 16 и 17, независимо от того, являются ли они криптографически безопасными или нет .

Потоковые шифры в режиме счетчика

У потоковых шифров в режиме счетчика простые функции следующего состояния и сложные функции вых ода, зависящие от ключа. Этот метод, показанный на -5-й, был предложен в [498, 715]. Функция следующего состояния может быть чем-то простым, например, счетчиком, добавляющим единицу к предыдущему состо янию.

Рис. 9-14. Генератор потока ключей в режиме счетчика.

Потоковый шифр в режиме счетчика может генерировать *i*-ый бит, k_i, без выдачи всех предшествующих ключевых битов. Просто установите счетчик вручную в *i*-ое внутреннее состояние и генерируйте бит. Это полезно для закрытия файлов данных с произвольным доступом, можно расшифровать конкретный блок данных не расшифровывая целый файл.

9.10 Другие режимы блочных шифров

Режим сцепления блоков

Для использования блочного алгоритма в режиме **сцепления блоков** (block chaining, BC), просто выполните XOR входа блочного шифра и результата XOR всех предыдущих блоков шифротекста. Как и для CBC используется IV. Математически это выглядит как:

C, = *Ek(P, Q F*; F,* I = *F, © C, P, = F, ©* *(C,); *Fi** I = *F,*

© Ci

Как и CBC, обратная связь процесса BC приводит к распространению ошибки в открытом тексте. Главная

проблема ВС заключается в том, что из-за того, что дешифрирование блока шифротекста зависит от всех пр едыдущих блоков шифротекста, единственная ошибка шифротекста приведет к неправильной расшифровке всех последующих блоков шифротекста.

Режим распространяющегося сцепления блоков шифра

Режим **распространяющегося сцепления блоков шифра** (propagating cipher block chaining, PCBC) [1080] похож на режим CBC за исключением того, что и предыдущий блок открытого текста, и предыдущий блок шифротекста подвергаются операции XOR с текущим блоком открытого текста перед шифрованием (или после шифрования) (см. -6-й).

$$Ci = E*P, \odot Ci \text{ I} \odot P, \text{ I)} \quad P* = Cj \text{ I} \odot Pi \text{ I} \odot \text{a*,)}$$

PCBC используется в Kerberos версии 4 (см. раздел 24.5) для выполнения за один проход и шифрования, и проверки целостности. В режиме PCBC ошибка шифротекста приводит к неправильному дешифрированию всех последующих блоков. Это означает, что проверка стандартного блока в конце сообщения обеспечивает целос тность всего сообщения.

Рис. 9-15. Режим распространяющегося сцепления блоков шифра.

К несчастью в этом режиме существует одна проблема [875]. Перестановка двух блоков шифротекста прив одит к неправильной расшифровке двух соответствующих блоков открытого текста, но из-за природы операции XOR над открытым текстом и шифротекстом, дальнейшие ошибки компенсируются. Поэтому, если при провер ке целостности проверяются только несколько последних блоков расшифрованного открытого текста, можно получить частично испорченное сообщение. Хотя никто до сих пор не додумался, как воспользоваться этой сл абостью, Kerberos версии 5 после обнаружения ошибки переключается в режим CBC.

Сцепление блоков шифра с контрольной суммой

Сцепление блоков шифра с контрольной суммой (cipher block chaining with checksum, CBCC) представ ляет собой вариант CBC [1618]. Сохраняйте значение XOR всех уже зашифрованных блоков открытого текста, выполняя для каждого текущего блока открытого текста перед его шифрованием XOR с сохраняемым значен ием. CBCC обеспечивает, что любое изменение любого блока шифротекста изменит результат дешифровки п оследнего блока. Если последний блок содержит какую-нибудь константу или служит для проверки целостности, то целостность расшифрованного открытого текста может быть проверена с минимальными дополнительными накладными расходами.

Выходная обратная связь с нелинейной функцией

Выходная обратная связь с нелинейной функцией (output feedback with a nonlinear function, OFBNLF) [777] представляет собой вариант и OFB, и ECB, где ключ изменяется с каждым блоком:

$$\text{C, } = Ek*P*, \; K* = Edit, \; \text{,1 } P, = \text{a*,}); \; Ki = E*K, \text{ I)}$$

Ошибка одного бита шифротекста распространяется только на один блок открытого текста. Однако, если бит теряется или добавляется, то ошибка распространяется до бесконечности. С блочным алгоритмом, использу ющим сложный алгоритм планирования ключей, этот режим работает медленно. Я не знаю, как выполнять кри птоанализ этого режима.

Прочие режимы

Возможны и другие режимы, хотя они используются нечасто. Сцепление блоков открытого текста (plaintext block chaining, PBC) похоже на CBC за исключением того, что операция XOR выполняется для с блока откр ытого текста и для предыдущего блока открытого текста, а не блока шифротекста. Обратная связь по открытому тексту (plaintext feedback, PFB) похожа на CFB за исключением того, что для обратной связи используется не шифротекст, а открытый текст. Существует также сцепление блоков шифротекста по различиям открытого те кста (cipher block chaining of plaintext difference, CBCPD). Я уверен, что можно найти еще таинственнее.

Если у криптоаналитика есть машина для поиска ключей грубой силой, то он сможет раскрыть ключ, если угадает один из блоков открытого текста. Некоторые из упомянутых странных режимов, по сути, являются д ополнительным шифрованием перед использованием алгоритма шифрования: например, XOR текста и фиксиро ванной секретной строки или перестановка текста. Почти все отклонения от стандартов помешают подобному криптоанализу.

9.11 Выбор режима шифра

Если вашей основной заботой являются скорость и простота, то ECB является самым простым и самым быстрым способом использовать блочный шифр. Помимо уязвимости к вскрытию повтором, алгоритм в режиме ECB проще всего криптоанализировать. Я не советую использовать ECB для шифрования сообщений.

ECB хорошо использовать для шифрования случайных данных, например, других ключей. Так как данные невелики по размеру и случайны, недостатки ECB не существенны для такого применения.

Для обычного открытого текста используйте CBC, CFB или OFB. Конкретный режим зависит от ваших требований. В приведены безопасность и эффективность различных режимов.

Для шифрования файлов лучше всего подходит CBC. Значительно увеличивается безопасность, и при появлении ошибок в хранимых данных почти никогда не бывает сбоев синхронизации. Если ваше приложение - программное, то CBC почти всегда будет лучшим выбором.

Табл. 9-1.
Краткий обзор режимов работы блочных шифров

ECB:

Security:

-Plaintext patterns are not concealed.

- Input to the block cipher Is not randomlzed; It Is the same as the plaintext. + More than one message can be encrypted with the same

- plaintext Is easy to manipulate; blocks can be removed, repeated, or Interchanged.

Efficiency: + Speed is the same as the block cipher.

- Clphertext Is up to one block longer than the plaintext, due to padding.

- No preprocessing is possible. *Processing is paraUelizable.

Fault-tolerance:

-A ciphertext error affects one full block of plaintext.

- Synchronization error is unrecoverable.

CFB:

Security:

+ Plaintext patterns are concealed. + Input to the block cipher is randomized. + More than one message can be encrypted with the same key, provided that a different IV is used. +/- Plaintext is somewhat difficult to manipulate; blocks call be removed from the beginning and end of the message, bits of the first block can be changed, and repetition allows some controlled changes.

Efficiency: + Speed is the same as the block cipher.

- Ciphertext is the same size as the plaintext, not counting the IV.

+/- Encryption is not paraUelizable; decryption is paral- Idizable and has a random-access property.

- Some preprocessing is possible before a block is seen; the Previous ciphertext block can be encrypted. +/- Encryption is not parallelizable; decry p-tion is paral- felizable and has a random-access property.

F'auh-toterance:

-A ciphertext error affects the corresponding bit of plaintext and the next full block.

+ Synchronization errors of full block sizes are recoverable. I. -bit CFB can recover from the addition or loss of single bits.

cbc:

Security:

+ Plaintext patterns are concealed by XORing with previous ciphertext block.

+ Input to the block cipher is randomized by XORing with the previous ciphertext block.

+ More than one message can be encrypted with the same key.

+/- Plaintext is somewhat difficult to manipulate; blocks can be removed from the beginning and end of the message, bits of the first block can be changed, and repetition allows some controlled changes.

Efficiency: + Speed is the same as the block cipher.

- Ciphertext is up to one block longer than the plaintext, not counting the IV.

- No preprocessing is possible.

+/- Encryption is not paraUelizable; decryption is paral- lelizable and has a random-access property.

Wau*-toterance:

- A ciphertext error affects one full block of plaintext and the corresponding bit in the next block.

- Synchronization error is unrecoverable.

OFB/Counter:

Security;

+ Plaintext patterns are concealed. + Input to the block cipher is randomized. + More than one message can be encrypted with the same key, provided that a different IV is used. - Plaintext is very easy to manipulate; any change in ciphertext directly affects the plaintext.

C*lclency: + Speed is the same as the block cipher.

- Ciphertext is the same size as the plaintext, not counting the IV. + Processing is possible before the message is seen.

-/+ OFB processing is not paraUelizable; counter processing is paraUelizable.

Fau*t-tolerance:

+ A ciphertext error affects only the corresponding bit of plaintext. - Synchronization error is unrecoverable.

CFB-specifically 8-bit CFB-is generally the mode *ol* choice for encrypting streams of characters when each cha r-acter has to be treated individually, as in a link between a terminal and a host. OFB is most often used in high-speed synchronous systems where error propagation is intolerable. OFB is also the mode of choice if preprocessing is r e-quired.

OFB is the mode of choice in a error-prone environment, because it has no error extension.

Stay away from the weird modes. One of the four basic modes-ECB, CBC, OFB, and CFB-is suitable for almost any application. These modes are not overly complex and probably do not reduce the security of the system. While it is possible that a complicated mode might increase the security of a system, most likely it just increases the complexity. None of the weird modes has any better error propagation or error recovery characteristics.

9.12 INTERLEAVING

With most modes, encryption of a bit (or block) depends on the encryption of the previous bits (or blocks). This can often make it impossible to parallelize encryption. For example, consider a hardware box that does encryption in CBC mode. Even if the box contains four encryption chips, only one can work at any time. The next chip needs the results of the previous chip before it starts working.

The solution is to **interleave** multiple encryption streams. (This is not multiple encryption; that's covered in Se c-tions 15.1 and 15.2). Instead of a single CBC chain, use four. The first, fifth, and every fourth block thereafter are e n-crypted in CBC mode with one IV. The second, sixth, and every fourth block thereafter are encrypted in CBC mode with another IV, and so on. The total IV is much longer than it would have been without interleaving.

Think of it as encrypting four different messages with the same key and four different IVs. These messages are all i nterleaved.

This trick can also be used to increase the overall speed of hardware encryption. If you have three encryption chips, each c a-pable of encrypting data at 33 megabits/second, you can interleave them to encrypt a single 100 megabit/second data channel.

Figure 9.16 shows three parallel streams interleaved in CFB mode. The idea can also work in CBC and OFB modes, and with any number of parallel streams. Just remember that each stream needs its own IV. Don't share.

9.13 BLOCK CIPHERS VERSUS STREAM CIPHERS

Although block and stream ciphers are very different, block ciphers can be implemented as stream ciphers and stream ciphers can be implemented as block ciphers. The best definition of the difference I've found is from Ranier Rueppel [1362.]:

> Block ciphers operate on data with a fixed transformation on large blocks of plaintext data; stream ciphers ope r-ate with a time-varying transformation on individual plaintext digits.

Figure 9.16 Interleavingthtee CFB encryptions.

In the real world, block ciphers seem to be more general (i.e., they can be used in any of the four modes) and stream ciphers seem to be easier to analyze mathematically. There is a large body of theoretical work on the analysis and design of stream c i-phers-most of it done in Europe, for some reason. They have been used by the world's militaries since the invention of electronics. This seems to be changing; recently a whole slew of theoretical papers have been written on block cipher design. Maybe soon there will be a theory of block cipher design as rich as our current theory of stream cipher d esign.

Otherwise, the differences between stream ciphers and block ciphers are in the implementation. Stream ciphers that only e n-crypt and decrypt data one bit at a time are not really suitable for software implementation. Block ciphers can be easier to impl e-ment in software, because they often avoid time-consuming bit manipulations and they operate on data in computer-sized blocks. On the other hand, stream ciphers can be more suitable for hardware implementation because they can be implemented very eff i-ciently in silicon.

These are important considerations. It makes sense for a hardware encryption device on a digital communications channel to encrypt the individual bits as they go by. This is what the device sees. On the other hand, it makes no sense for a software encry p-tion device to encrypt each individual bit separately. There are some specific instances where bit- and byte-wise encryption might be necessary in a computer system-encrypting the link between the keyboard and the CPU, for example-but generally the encry p-tion block should be at least the width of the data bus.

Глава 10 Using Algorithms

Think of security - data security, communications security, information security, whatever - as a chain. The security of the entire system is only as strong as the weakest link. Everything has to be secure: cryptographic algorithms, protocols, key manag e-ment, and more. If your algorithms are great but your random-number generator stinks, any smart cryptanalyst is going to attack your system through the random-number generation. If you patch that hole but forget to securely erase a memory location that contains the key, a cryptanalyst will break your system via that route. If you do everything right and accidentally e-mail a copy of your secure files to The Wall Street Journal, you might as well not have bothered.

It's not fair. As the designer of a secure system, you have to think of every possible means of attack and protect against them all, but a cryptanalyst only has to find one hole in your security and exploit it.

Cryptography is only a part of security, and often a very small part. It is the mathematics of making a system secure, which is different from actually making a system secure. Cryptography has its "size queens": people who spend so much time arguing about how long a key should be that they forget about everything else. If the secret police want to know what is on your computer, it is far easier for them to break into your house and install a camera that can record what is on your computer screen than it is for them to cryptanalyze your hard drive.

Additionally, the traditional view of computer cryptography as "spy versus spy" technology is becoming increasingly ina p-propriate. Over 99 percent of the cryptography used in the world is not protecting military secrets; it's in applications such as bank cards, pay-TV, road tolls, office building and computer access tokens, lottery terminals, and prepayment electricity meters [43,44]. In these applications, the role of cryptography is to make petty crime slightly more difficult; the paradigm of the well-funded a d-versary with a rabbit warren of cryptanalysts and roomsful of computers just doesn't apply.

Most of those applications have used lousy cryptography, but successful attacks against them had nothing to do with cry p-tanalysts. They involved crooked employees, clever sting operations, stupid implementations, integration blunders, and random idiocies. (I strongly recommend Ross Anderson's paper, "Why Cryptosytems Fail" [44]; it should be required reading for anyone involved in this field.) Even the NSA has admitted that most security failures in its area of interest are due to failures in impl e-mentation, and not failures in algorithms or protocols [1119]. In these instances it didn't matter how good the cryptography was; the successful attacks bypassed it completely.

10.1 CHOOSING AN ALGORITHM

When it comes to evaluating and choosing algorithms, people have several alternatives:

- They can choose a published algorithm, based on the belief that a published algorithm has been scrutinized by many cry p-tographers; if no one has broken the algorithm yet, then it must be pretty good.

- They can trust a manufacturer, based on the belief that a well-known manufacturer has a reputation to uphold and is u n-likely to risk that reputation by selling equipment or programs with inferior algorithms.

- They can trust a private consultant, based on the belief that an impartial consultant is best equipped to make a reliable evaluation of different algorithms.

- They can trust the government, based on the belief that the government is trustworthy and wouldn't steer its citizens wrong.

- They can write their own algorithms, based on the belief that their cryptographic ability is second-to-none and that they should trust nobody but themselves.

Any of these alternatives is problematic, but the first seems to be the most sensible. Putting your trust in a single manufa c-turer, consultant, or government is asking for trouble. Most people who call themselves security consultants (even those from big-name firms usually don't know anything about encryption. Most security product manufacturers are no better. The NSA has some of the world's best cryptographers working for it, but they're not telling all they know. They have their own interests to further which are not congruent with those of their citizens. And even if you're a genius, writing your own algorithm and then using it without any peer review is just plain foolish.

The algorithms in this book are public. Most have appeared in the open literature and many have been cryptanalyzed by e x-perts in the field. I list all published results, both positive and negative. I don't have access to the cryptanalysts done by any of the myriad military security organizations in the world Which are probably better than the academic institutions,(they've been doing it longer and are better funded), so it is possible that these algorithms are easier to break than it appears. Even so, it is far more likely that they are more secure than an algorithm designed and implemented in secret in some corporate basement.

The hole in all this reasoning is that we don't know the abilities of the various military cryptanalysts organizations.

What algorithms can the NSA break? For the majority of us, there's really no way of knowing. If you are arrested with a DES-encrypted computer hard drive, the FBI is unlikely to introduce the decrypted plaintext at your trial; the fact that they can break an algorithm is often a bigger secret than any information that is recovered. During WWII, the Allies were forbidden from using decrypted German Ultra traffic unless they could have plausibly gotten the information elsewhere. The only way to get the NSA to admit to the ability to break a given algorithm is to encrypt something so valuable that its public dissemination is worth the admission. Or, better yet, create a really funny joke and send it via encrypted e-mail to shady characters in shadowy countries. NSA employees are people, too; I doubt even they can keep a good joke secret.

A good working assumption is that the NSA can read any message that it chooses, but that it cannot read all messages that it chooses. The NSA is limited by resources, and has to pick and choose among its various targets. Another good assumption is that they prefer breaking knuckles to breaking codes; this preference is so strong that they will only resort to breaking codes when they wish to preserve the secret that they have read the message. In any case, the best most of us can do is to choose among public a l-gorithms that have withstood a reasonable amount of public scrutiny and cryptanalysts. Algorithms for Export

Algorithms for export out of the United States must be approved by the U.S. government (actually, by the NSA (see Section 25.1). It is widely believed that these export-approved algorithms can be broken by the NSA. Although no one has admitted this on the record, these are some of the things the NSA is rumored to privately suggest to companies wishing to export their crypt o-graphic products:

- Leak a key bit once in a while, embedded in the ciphertext.

- "Dumb down" the effective key to something in the 30-bit range. For example, while the algorithm might accept a 100-bit key, most of those keys might be equivalent.

- Use a fixed IV, or encrypt a fixed header at the beginning of each encrypted message. This facilitates a known-plaintext attack.

- Generate a few random bytes, encrypt them with the key, and then put both the plaintext and the ciphertext of those ra n-dom bytes at the beginning of the encrypted message. This also facilitates a known- plaintext attack.

NSA gets a copy of the source code, but the algorithm's details remain secret from everyone else. Certainly no one adve r-tises any of these deliberate weaknesses, but beware if you buy a U.S. encryption product that has been approved for export.

10.2 PUBLIC-KEY CRYPTOGRAPHY VERSUS SYMMETRIC CRYPTOGRAPHY

Which is better, public-key cryptography or symmetric cryptography? This question doesn't make any sense, but has been d e-bated since public-key cryptography was invented. The debate assumes that the two types of cryptography can be compared on an equal footing. They can't.

Needham and Schroeder [1159] pointed out that the number and length of messages are far greater with public-key alg o-rithms than with symmetric algorithms. Their conclusion was that the symmetric algorithm was more efficient than the public-key algorithm. While true, this analysis overlooks the significant security benefits of public-key cryptography. Whitfield Diffie writes 492,494]:

In viewing public-key cryptography as a new form of cryptosystem rather than a new form of key management, I set the stage for criticism on grounds of both security and performance. Opponents were quick to point out that the RSA system ran about one-thousandth as fast as DES and required keys about ten times as large. Although it had been obvious from the beginning that the use of public key systems could be limited to exchanging keys for conventional [symmetric] cryptography, it was not immediately clear that this was necessary. In this context, the proposal to build hybrid systems [879] was hailed as a discovery in its own right.

Public-key cryptography and symmetric cryptography are different sorts of animals; they solve different sorts of problems. Symmetric cryptography is best for encrypting data. It is orders of magnitude faster and is not susceptible to chosen-ciphertext a t-tacks. Public-key cryptography can do things that symmetric cryptography can't; it is best for key management and a myriad of protocols discussed in Part I.

Other primitives were discussed in Part I: one-way hash functions, message authentication codes, and so on. Table 10.1 lists different types of algorithms and their properties [804].

10.3 ENCRYPTING COMMUN1CAT10NS CHANNELS

This is the classic Alice and Bob problem: Alice wants to send Bob a secure message. What does she do? She encrypts the message.

In theory, this encryption can take place at any layer in the OSI (Open Systems Interconnect) communications model. (See the OSI security architecture standard for more information [305].) In practice, it takes place either at the lowest layers (one and two) or at higher layers. If it takes place at the lowest layers, it is called link-by-link encryption; everything going through a particular data link is encrypted. If it takes place at higher layers, it is called end-to-end encryption; the data are encrypted selectively and stay encrypted until they are decrypted by the intended final recipient. Each approach has its own benefits and drawbacks.

Link-by Link Encryption

The easiest place to add encryption is at the physical layer (see Figure 10. 1). This is called link-by-link encryption. The interfaces to the physical layer are generally standardized and it is easy to connect hardware encryption devices at this point. These devices encrypt all data passing through them, including data, routing information, and protocol information. They can be used on any type of digital communication link. On the other hand, any intelligent switching or storing nodes between the sender and the receiver need to decrypt the data stream before processing it.

This type of encryption is very effective. Because everything is encrypted, a crypt- analyst can get no information about the structure of the information. He has no idea who is talking to whom, how long the messages they are sending are, what times of day they communicate, and so on. This is called traffic-flow security: the enemy is not only denied access to the information, but also access to the knowledge of where and how much information is flowing.

Security does not depend on any traffic management techniques. Key management is also simple; only the two endpoints of the line need a common key, and they can change their key independently from the rest of the network.

Imagine a synchronous communications line, encrypted using 1-bit CFB. After initialization, the line can run indefinitely, re-

covering automatically from bit or synchronization errors. The line encrypts whenever messages are sent from one end to the other; otherwise it just encrypts and decrypts random data. Eve has no idea when messages are being sent and when they are not; she has no idea when messages begin and end. All she sees is an endless stream of random-looking bits.

If the communications line is asynchronous, the same 1-bit CFB mode can be used. The difference is that the adversary can get information about the rate of transmission. If this information must be concealed, make some provision for passing dummy messages during idle times.

The biggest problem with encryption at the physical layer is that each physical link in the network needs to be encrypted: Leaving any link unencrypted jeopardizes the security of the entire network. If the network is large, the cost may quickly become prohibitive for this kind of encryption.

Additionally, every node in the network must be protected, since it processes unencrypted data. If all the network's users trust one another, and all nodes are in secure locations, this may be tolerable. But this is unlikely. Even in a single corporation, information might have to be kept secret within a department. If the network accidentally misroutes information, anyone can read it. Table 10.2 summarizes the pros and cons of link-by-link encryption.

End-to-End Encryption

Another approach is to put encryption equipment between the network layer and the transport layer. The encryption device must understand the data according to the protocols up to layer three and encrypt only the transport data units, which are then r e-combined with the unencrypted routing information and sent to lower layers for transmission.

This approach avoids the encryption/decryption problem at the physical layer. By providing end-to-end encryption, the data remains encrypted until it reaches its final destination (see Figure 10.2). The primary problem with end-to-end encryption is that the routing information for the data is not encrypted; a good cryptanalyst can learn much from who is talking to whom, at what times and for how long, without ever knowing the contents of those conversations. Key management is also more difficult, since individual users must make sure they have common keys.

Building end-to-end encryption equipment is difficult. Each particular communications system has its own protocols. Som e-times the interfaces between the levels are not well-defined, making the task even more difficult.

If encryption takes place at a high layer of the communications architecture, like the applications layer or the presentation layer, then it can be independent of the type of communication network used. It is still end-to-end encryption, but the encryption implementation does not have to bother about line codes, synchronization between modems, physical interfaces, and so forth. In the early days of electro- mechanical cryptography, encryption and decryption took place entirely offline; this is only one step r e-moved from that.

Encryption at these high layers interacts with the user software. This software is different for different computer archite c-tures, and so the encryption must be optimized for different computer systems. Encryption can occur in the software itself or in specialized hardware. In the latter case, the computer will send the data to the specialized hardware for encryption before sending it to lower layers of the communication architecture for transmission. This process requires some intelligence and is not suitable for dumb terminals. Additionally, there may be compatibility problems with different types of computers. The major disadvantage of end-to-end encryption is that it allows traffic analysis. Traffic analysis is the analysis of encrypted messages: where they come from, where they go to, how long they are, when they are sent, how frequent or infrequent they are, whether they coincide with outside events like meetings, and more. A lot of good information is buried in that data, and a cryptanalyst will want to get his hands on it. Table 10.3 presents the positive and negative aspects of end-to-end encryption.

Combining the Two

Table 10.4, primarily from [1244], compares link-by-link and end-to-end encryption. Combining the two, while most expe n-sive, is the most effective way of securing a network. Encryption of each physical link makes any analysis of the routing inform a-tion impossible, while end-to-end encryption reduces the threat of unencrypted data at the various nodes in the network. Key ma n-agement for the two schemes can be completely separate: The network managers can take care of encryption at the physical level, while the individual users have responsibility for end-to-end encryption.

10.4 ENCRYPTING DATA FOR STORAGE

Encrypting data for storage and later retrieval can also be thought of in the Alice and Bob model. Alice is still sending a me s-sage to Bob, but in this case "Bob" is Alice at some future time. However, the problem is fundamentally different. In communic a-tions channels, messages in transit have no intrinsic value. If Bob doesn't receive a particular message, Alice can always resend it. This is not true for data encrypted for storage. If Alice can't decrypt her message, she can't go back in time and re-encrypt it. She has lost it forever. This means that encryption applications for data storage should have some mechanisms to prevent unrecove r-able errors from creeping into the ciphertext. The encryption key has the same value as the message, only it is smaller. In effect, cryptography converts large secrets into smaller ones. Being smaller, they can be easily lost. Key management procedures should assume that the same keys will be used again and again, and that data may sit on a disk for years before being decrypted. Fu r-thermore, the keys will be around for a long time. A key used on a communications link should, ideally, exist only for the length of the communication. A key used for data storage might be needed for years, and hence must be stored securely for years.

Other problems particular to encrypting computer data for storage were listed in [357]:

- The data may also exist in plaintext form, either on another disk, in another computer, or on paper. There is much more opportunity for a cryptanalyst to perform a known-plaintext attack.

- In database applications, pieces of data may be smaller than the block size of most algorithms. This will cause the ciphe r-text to be considerably larger than the plaintext.

- The speed of I/O devices demands fast encryption and decryption, and will probably require encryption hardware. In some applications, special high-speed algorithms may be required.

- Safe, long-term storage for keys is required.

- Key management is much more complicated, since different people need access to different files, different portions of the same file, and so forth. If the encrypted files are not structured as records and fields, such as text files, retrieval is easier: The entire file is decrypted before use. If the encrypted files are database files, this solution is problematic. Decrypting the entire dat a-base to access a single record is inefficient, but encrypting records independently might be susceptible to a block-replay kind of attack. In addition, you must make sure the unencrypted file is erased after encryption (see Section 10.9). For further details and insights, consult [425,569].

Dereferencing Keys

When encrypting a large hard drive, you have two options. You can encrypt all the data using a single key. This gives a cryptanalyst a large amount of ciphertext to analyze and makes it impossible to allow multiple users to see only parts of the drive. Or, you can encrypt each file with a different key, forcing users to memorize a different key for each file.

The solution is to encrypt each file with a separate key, and to encrypt the keys with another key known by the users. Each user only has to remember that one key. Different users can have different subsets of the file-encryption keys encrypted with their key. And there can even be a master key under which every file-encryption key is encrypted. This is even more secure because the file-encryption keys are random and less susceptible to a dictionary attack.

Driver-Level vs. File-Level Encryption

There are two ways to encrypt a hard drive: at the file level and at the driver level. Encryption at the file level means that every file is encrypted separately. To use a file that's been encrypted, you must first decrypt the file, then use it, and then re- e n-crypt it.

Driver-level encryption maintains a logical drive on the user's machine that has all data on it encrypted. If done well, this can provide security that, beyond choosing good passwords, requires little worry on the part of the user. The driver must be consider a-bly more complex than a simple file-encryption program, however, because it must deal with the issues of being an installed d e-vice driver, allocation of new sectors to files, recycling of old sectors from files, random-access read and update requests for any data on the logical disk, and so on.

Typically, the driver prompts the user for a password before starting up. This is used to generate the master decryption key, which may then be used to decrypt actual decryption keys used on different data.

Providing Random Access to an Encrypted Drive

Most systems expect to be able to access individual disk sectors randomly. This adds some complication for using many stream ciphers and block ciphers in any chaining mode. Several solutions are possible.

Use the sector address to generate a unique IV for each sector being encrypted or decrypted. The drawback is that each se c-tor will always be encrypted with the same IV. Make sure this is not a security problem.

For the master key, generate a pseudo-random block as large as one sector. You can do this by running an algorithm in OFB mode, for example.) To encrypt any sec- tor, first XOR in this pseudo-random block, then encrypt normally with a block cipher in ECB mode. This is called ECB+OFB (see Section 15.4).

Since CBC and CFB are error-recovering modes, you can use all but the first block or two in the sector to generate the IV for that sector. For example, the IV for sector 3001 may be the hash of the all but the first 128 bits of the sector's data. After genera t-ing the IV, encrypt normally in CBC mode. To decrypt the sector, you use the second 64-bit block of the sector as an IV, and d e-crypt the remainder of the sector. Then, using the decrypted data, you regenerate the IV and decrypt the first 128 bits.

You can use a block cipher with a large enough block size that it can encrypt the whole sector at once. Crab See Section 14.6) is an example.

10.5 HARDWARE ENCRYPTION VERSUS SOFTWARE ENCRYPTION

Hardware

Until very recently, all encryption products were in the form of specialized hardware. These encryption/decryption boxes plugged into a communications line and encrypted all the data going across that line. Although software encryption is becoming more prevalent today, hardware is still the embodiment of choice for military and serious commercial applications. The NSA, for example, only authorizes encryption in hardware. There are several reasons why this is so.

The first is speed. As we will see in Part III, encryption algorithms consist of many complicated operations on plaintext bits. These are not the sorts of operations that are built into your run-of-the-mill computer. The two most common encryption algorithms, DES and RSA, run inefficiently on general-purpose processors. While some cryptographers have tried to make their algorithms more suitable for software implementation, specialized hardware will always win a speed race.

Additionally, encryption is often a computation-intensive task. Tying up the computer's primary processor for this is inefficient. Moving encryption to another chip, even if that chip is just another processor, makes the whole system faster. The second reason is security. An encryption algorithm running on a generalized computer has no physical protection. Mallory can go in with various debugging tools and surreptitiously modify the algorithm without anyone ever realizing it. Hardware encryption devices can be securely encapsulated to prevent this. Tamper- proof boxes can prevent someone from modifying a hardware encryption device. Special-purpose VLSI chips can be coated with a chemical such that any attempt to access their interior will result in the destruction of the chip's logic. The U.S. government's Clipper and Capstone chips See Sections 24.16 and 24.171 are designed to be tamperproof. The chips can be designed so that it is impossible for Mallory to read the unencrypted key.

IBM developed a cryptographic system for encrypting data and communications on mainframe computers [515,1027]. It includes tamper-resistant modules to hold keys. This system is discussed in Section 24.1.

Electromagnetic radiation can sometimes reveal what is going on inside a piece of electronic equipment. Dedicated encryption boxes can be shielded, so that they leak no compromising information. General-purpose computers can be shielded as well, but it is a far more complex problem. The U.S. military calls this TEMPEST; it's a subject well beyond the scope of this book.

The final reason for the prevalence of hardware is the ease of installation. Most encryption applications don't involve general-purpose computers. People may wish to encrypt their telephone conversations, facsimile transmissions, or data links. It is cheaper to put special-purpose encryption hardware in the telephones, facsimile machines, and modems than it is to put in a microprocessor and software.

Even when the encrypted data comes from a computer, it is easier to install a dedicated hardware encryption device than it is to modify the computer's system software. Encryption should be invisible; it should not hamper the user. The only way to do this in software is to write encryption deep into the operating system. This isn't easy. On the other hand, even a computer neophyte can plug an encryption box between his computer and his external modem.

The three basic kinds of encryption hardware on the market today are: self-contained encryption modules (that perform functions such as password verification and key management for banks), dedicated encryption boxes for communications links, and boards that plug into personal computers.

Some encryption boxes are designed for certain types of communications links, such as T-1 encryption boxes that are designed not to encrypt synchronization bits. There are different boxes for synchronous and asynchronous communications lines. Newer boxes tend to accept higher bit rates and are more versatile.

Even so, many of these devices have some incompatibilities. Buyers should be aware of this and be well-versed in their par-

ticular needs, lest they find themselves the owners of encryption equipment unable to perform the task at hand. Pay attention to restrictions in hardware type, operating system, applications software, net- work, and so forth. PC-board encryptors usually e n-crypt everything written to the hard disk and can be configured to encrypt everything sent to the floppy disk and serial port as well. These boards are not shielded against electromagnetic radiation or physical interference, since there would be no benefit in pr o-tecting the boards if the computer remained unaffected. More companies are starting to put encryption hardware into their co m-munications equipment. Secure telephones, facsimile machines, and modems are all available. Internal key management for these devices is generally secure, although there are as many different schemes as there are equipment vendors. Some schemes are more suited for one situation than another, and buyers should know what kind of key management is incorporated into the encryption box and what they are expected to provide themselves.

Software

Any encryption algorithm can be implemented in software. The disadvantages are in speed, cost, and ease of modification (or manipulation). The advantages are in flexibility and portability, ease of use, and ease of upgrade. The algorithms written in C at the end of this book can be implemented, with little modification, on any computer. They can be inexpensively copied and i n-stalled on many machines. They can be incorporated into larger applications, such as communications programs or word proce s-sors.

Software encryption programs are popular and are available for all major operating systems. These are meant to protect i n-dividual files; the user generally has to manually encrypt and decrypt specific files. It is important that the key management scheme be secure: The keys should not be stored on disk anywhere (or even written to a place in memory from where the processor swaps out to disk). Keys and unencrypted files should be erased after encryption. Many programs are sloppy in this regard, and a user has to choose carefully.

Of course, Mallory can always replace the software encryption algorithm with something lousy. But for most users, that isn't a problem. If Mallory can break into our office and modify our encryption program, he can also put a hidden camera on the wall, a wiretap on the telephone, and a TEMPEST detector down the street. If Mallory is that much more powerful than the user, the user has lost the game before it starts.

10.6 COMPRESSION, ENCODING, AND ENCRYPTION

Using a data compression algorithm together with an encryption algorithm makes sense for two reasons:

Cryptanalysis relies on exploiting redundancies in the plaintext; com- pressing a file before encryption reduces these redu n-dancies.

Encryption is time-consuming; compressing a file before encryption speeds up the entire process.

The important thing to remember is to compress before encryption. If the encryption algorithm is any good, the ciphertext will not be compressible; it will look like random data. (This makes a reasonable test of an encryption algorithm; if the cipher-text can be compressed, then the algorithm probably isn't very good.)

If you are going to add any type of transmission encoding or error detection and recovery, remember to add that after encry p-tion. If there is noise in the communications path, decryption's error-extension properties will only make that noise worse. Figure 10.3 summarizes these steps.

10.7 DETECTING ENCRYPTION

How does Eve detect an encrypted file? Eve is in the spy business, so this is an important question. Imagine that she's eave s-dropping on a network where messages are flying in all directions at high speeds; she has to pick out the interesting ones. E n-crypted files are certainly interesting, but how does she know they are encrypted?

Generally, she relies on the fact that most popular encryption programs have well-defined headers. Electronic-mail messages encrypted with either PEM or POP (see Sections 24.10 and 24.12) are easy to identify for that reason.

Other file encryptors just produce a ciphertext file of seemingly random bits. How can she distinguish it from any other file of seemingly random bits? There is no sure way, but Eve can try a number of things:

- Examine the file. ASCII text is easy to spot. Other file formats, such as TIFF, TeX, C, Postscript, G3 facsimile, or Microsoft Excel, have standard identifying characteristics. Executable code is detectable, as well. UNIX files often have "magic numbers" that can be detected.

- Try to uncompress the file, using the major compression algorithms. If the file is compressed (and not encrypted), this should yield the original file.

- Try to compress the file. If the file is ciphertext (and the algorithm is good), then the probability that the file can be appreciably compressed by a general-purpose compression routine is small. (By appreciably, I mean more than 1 or 2 percent.) If it is something else (a binary image or a binary data file, for examples it probably can be compressed.

Any file that cannot be compressed and is not already compressed is probably ciphertext. (Of course, it is possible to specifically make ciphertext that is compressible.) Identifying the algorithm is a whole lot harder. If the algorithm is good, you can't. If the algorithm has some slight biases, it might be possible to recognize those biases in the file. However, the biases have to be pretty significant or the file has to be pretty big in order for this to work.

10.8 HIDING CIPHERTEXT IN CIPHERTEXT

Alice and Bob have been sending encrypted messages to each other for the past year. Eve has been collecting them all, but she cannot decrypt any of them. Finally, the secret police tire of all this unreadable ciphertext and arrest the pair. "Give us your encryption keys," they demand. Alice and Bob refuse, but then they notice the thumbscrews. What can they do?

Wouldn't it be nice to be able to encrypt a file such that there are two possible decryptions, each with a different key. Alice could encrypt a real message to Bob in one of the keys and some innocuous message in the other key. If Alice were caught, she could surrender the key to the innocuous message and keep the real key secret.

The easiest way to do this is with one-time pads. Let P be the plaintext, D the dummy plaintext, C the ciphertext, K the real key, and K' the dummy key. Alice encrypts P:

$$P \oplus K = C$$

Alice and Bob share K, so Bob can decrypt C:

$$C \oplus K = P$$

If the secret police ever force them to surrender their key, they don't surrender K, but instead surrender:

$$K' = C \oplus D$$

The police then recover the dummy plaintext:

$$C \oplus K' = D$$

Since these are one-time pads and K is completely random, there is no way to prove that K' was not the real key. To make

matters more convincing, Alice and Bob should concoct some mildly incriminating dummy messages to take the place of the really incriminating real messages. A pair of Israeli spies once did this.

Alice could take P and encrypt it with her favorite algorithm and key K to get C. Then she takes C and XORs it with some piece of mundane plaintext - Pride and Prejudice for example, to get K'. She stores both C and the XOR on her hard disk. Now, when the secret police interrogate her, she can explain that she is an amateur cryptographer and that K' is a merely one-time pad for C. The secret police might suspect something, but unless they know K they cannot prove that Alice's explanation isn't valid.

Another method is to encrypt P with a symmetric algorithm and K, and D with K'. Intertwine bits (or bytes) of the ciphertext to make the final ciphertexts. If the secret police demand the key, Alice gives them K' and says that the alternating bits (or bytes) are random noise designed to frustrate cryptanalysts. The trouble is the explanation is so implausible that the secret police will probably not believe her (especially considering it is suggested in this book). A better way is for Alice to create a dummy me s-sage, D, such that the concatenation of P and D, compressed, is about the same size as D. Call this concatenation P'. Alice then encrypts P' with whatever algorithm she and Bob share to get C. Then she sends C to Bob. Bob decrypts C to get P', and then P and D. Then they both compute C 0 D = K'. This K' becomes the dummy one-time pad they use in case the secret police break their doors down. Alice has to transmit D so that hers and Bob's alibis match.

Another method is for Alice to take an innocuous message and run it through some error-correcting code. Then she can i n-troduce errors that correspond to the secret encrypted message. On the receiving end, Bob can extract the errors to reconstruct the secret message and decrypt it. He can also use the error-correcting code to recover the innocuous message. Alice and Bob might be hard pressed to explain to the secret police why they consistently get a 30 percent bit-error rate on an otherwise noise-free co m-puter network, but in some circumstances this scheme can work.

Finally, Alice and Bob can use the subliminal channels in their digital signature algorithms (see Sections 4.2 and 23.3). This is undetectable, works great, but has the drawback of only allowing 20 or so characters of subliminal text to be sent per signed innocuous message. It really isn't good for much more than sending keys.

10.9 DESTROYING INFORMATION

When you delete a file on most computers, the file isn't really deleted. The only thing deleted is an entry in the disk's index file, telling the machine that the file is there. Many software vendors have made a fortune selling file-recovery software that recovers files after they have been deleted.

And there's yet another worry: Virtual memory means your computer can read and write memory to disk any time. Even if you don't save it, you never know when a sensitive document you are working on is shipped off to disk. This means that even if you never save your plaintext data, your computer might do it for you. And driver-level compression programs like Stacker and DoubleSpace can make it even harder to predict how and where information is stored on a disk.

To erase a file so that file-recovery software cannot read it, you have to physically write over all of the file's bits on the disk. According to the National Computer Security Center [1148]:

> Overwriting is a process by which unclassified data are written to storage locations that previously held sensitive data.... To purge the ... storage media, the DoD requires overwriting with a pattern, then its complement, and finally with another pattern; e.g., overwrite first with 0011 0101, followed by 1100 1010, then 1001 0111. The number of times an overwrite must be acco m-plished depends on the storage media, sometimes on its sensitivity, and sometimes on different DoD component requirements. In any case, a purge is not complete until a final over- write is made using unclassified data.

You may have to erase files or you may have to erase entire drives. You should also erase all unused space on your hard disk.

Most commercial programs that claim to implement the DoD standard over- write three times: first with all ones, then with all zeros, and finally with a repeating one-zero pattern. Given my general level of paranoia, I recommend overwriting a deleted file seven times: the first time with all ones, the second time with all zeros, and five times with a cryptographically secure pseudo-random sequence. Recent developments at the National Institute of Standards and Technology with electron-tunneling microscopes suggest even that might not be enough. Honestly, if your data is sufficiently valuable, assume that it is impossible to erase data completely off magnetic media. Burn or shred the media; it's cheaper to buy media new than to lose your secrets.