CHAPMAN & HALL/CRC
CRYPTOGRAPHY AND NETWORK SECURITY

ВВЕДЕНИЕ В СОВРЕМЕННУЮ КРИПТОГРАФИЮ

Второе Издание

Джонатан Катц
Йехуда Линдел

CRC Press
Taylor & Francis Group
A CHAPMAN & HALL/CRC BOOK

Издательство Чэпмэн энд Холл /CRC
КРИПТОГРАФИЯ И СЕТЕВАЯ БЕЗОПАСНОСТЬ

ВВЕДЕНИЕ В СОВРЕМЕННУЮ КРИПТОГРАФИЮ

Издание второе

Джонатан Кац
Мэрилендский университет
Колледж Парк, шт. Мэриленд, США

Йехуда Линделл
Университет Бар-Илан,
г. Рамат-Ган, Израиль

Издательство «СиЭрСи»
«Тэйлор энд Фрэнсис Груп»
6000 Брокен Саунд Паркуэй Северо-Запад, Офис 300, Бока-Ратон, Флорида 33487-2742

Напечатано на бумаге, не содержащей кислот. Дата последней редакции: 20150318

Стандартный международный номер книги 13: 978-1-4822-5414-3 (Содержимое – Учебник и Электронная книга)

Библиотека Конгресса. Каталожные данные

Хэвилл, Джессен.

Изучение информатики: междисциплинарные задачи, основы и программирование на Python / автор Джессен Хэвилл.

страниц – («Чэпмэн энд Холл»/»СиЭрСи» учебники по информатике; 15), включая библиографические ссылки и алфавитный указатель.

ISBN 978-1-4822-5414-3 (бескислотная бумага)

1. Информатика-Учебники. 2. Python (язык программирования)--Учебники. I. Заголовок.

QA76.H3735 2015
005.13’3--dc23 2015004805

Посетите сайт «Тэйлор энд Фрэнсис» http://www.taylorandfrancis.com

и сайт Издательства «Си Эр Си» http://www.crcpress.com

Часть I

Вступление и классическая криптография

Глава 1

Введение

1.1 Криптография и современная криптография

В кратком словаре английского языка издательства Оксфорд (Concise Oxford English Dictionary) слово «криптография» определяется как «искусство составления или разгадывания шифров.” Хотя исторически точное, оно не передает всей широты данной дисциплины на текущем этапе или современных научных достижений в этой области. Данное определение всецело сосредотачивает внимание на шифрах, которые веками использовались для обеспечения тайного общения. Но сегодня охват криптографии гораздо шире: она рассматривает механизмы обеспечения целостности, методы обмена секретными ключами, протоколы аутентификации пользователей, электронные торги и выборы, электронные деньги и т.д. Без полного определния , мы можем сказать, что современная криптография включает в себя *изучение математических методов защиты цифровой информации, систем и распределенных вычислений от враждебных действий.*

Словарное определение, приведенное выше, также относит криптографию к одному из видов искусств. Пожалуй, до конца XX века она во многом оставалась искусством. В основе создания хороших шифров, как и вскрытия уже существующих, лежали творческий подход и глубокое понимание того, как работает шифр. Теоретическая база была скудной, да и в течение длительного времени не было действующего определения того, что же составляет хороший шифр. Коренным образом изменилось понимание криптографии в начале 70-х и 80-х гг. XX века . Стала нарабатываться теоретическая база, которая дала возможность скрупулезно изучать криптографию как науку и математическую дисциплину. В свою очередь данный аспект повлиял на то, как исследователи стали воспринимать более широкое поле компьютерной безопасности.

Другое важное различие между классической (скажем, до 1980-х гг.) и современной криптографией относится к области ее применения. Исторически основными пользователями «искусства шифрования» были военные организаций и правительства. Сегодня, криптография повсюду! Если В ы когда-либо проходили процедуру удостоверения путем ввода пароля, совершали покупки с помощью кредитной карты в Интернете или скачивали проверенное обновление для своей операционной системы, то несомненно, В ы пользовались криптографией. Все чаще программистов с относительно небольшим опытом работы

просят «защитить» приложения, которые они разрабатывают, путем внедрения механизмов криптографии.

Одним словом, криптография прошла путь от опытного набора инструментов, имеющих отношение к обеспечению секретности связи вооруженных сил, до науки, которая помогает защищать пользовательские системы по всему миру. Это, в свою очередь, означает, что криптография стала главной темой , рассматриваемой в рамках информатики.

Цели данной книги. Наша цель - донести основные принципы криптографии до аудитории студентов, изучающих информатику, электротехнику и электронику или математику, всех тех специалистов, кто желает добавить криптографию в разрабатываемые системы или программное обеспечение, а также всех, кто имеет необходимую математическую подготовку и проявляет интерес к освоению этой столь захватывающей области. После прочтения данной книги у читателя должно сформироваться понимание, тех гарантий защищенности, которые обеспечиваются базовыми криптографическими алгоритмами, знание стандартной структуры таких алгоритмов, а также базовые навыки оценки новых криптосистем на основе доказательств их стойкости (или отсутствии оной) и математических допущений, положенных в основу доказательств. У нас нет намерения сделать из читателей экспертов - или разработчиков новых криптосистем - по прочтении книги, но мы приложили усилия, чтобы ознакомить интересующихся с терминологическим аппаратом и фундаментальными знаниями, необходимыми для последующего изучения более специализированной литературы в данной области.

В этой главе. Центр внимания данной книги сосредоточен на формальном изучении современной криптографии, но в данной главе мы начинаем с неформального рассмотрения «классической» криптографии. Помимо подачи материала по принципу от простого к сложному, данный прием также служит цели мотивации изучения более строгого подхода, который используется в остальной части книги. Мы не стремимся здесь к доскональности, и потому данную главу не следует считать в качестве хронологической. Всем интересующимся историей криптографии рекомендуется обратиться к источникам, указанным в конце главы.

1.2 Параметры шифрования с закрытым ключом

Классическая криптография касалась создания и применения кодов (также называемые *шифры*), которые позволяют двум сторонам обмениваться информацией тайно даже при наличии подслушивающей третьей стороны, которая может отслеживать все процессы обмена данными. В современном языке коды называются *схемами шифрования*, и именно этот термин будет использоваться здесь. Надежность всех классических схем шифрования зависит

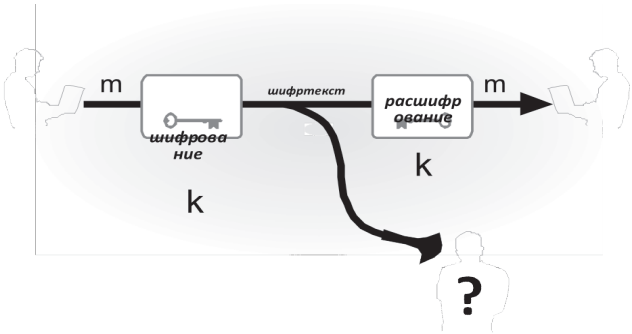

РИСУНОК 1.1: В закрытой криптосистеме (здесь, шифрование) используется один общий параметр: две стороны обмениваются одним ключом, которые они используют для защиты обмена информацией.

от секретной информации — ключа—заранее известной только общающимся сторонам, но которой не владеет подслушивающий. Такой вариант защиты известен как *шифрование (или симметрично -/секретным-ключом)* с закрытым ключом, и служит примером одного из многих криптографических алгоритмов, применяемого с таким параметром. Прежде чем приступить к описанию некоторых схем, ставших историей, рассмотрим симметричное шифрование (с закрытым ключом) более обобщенно.

В условиях шифрования с закрытым ключом две стороны используют один ключ, когда намерены тайно обменяться информацией. Одна сторона может отправить сообщение или *открытый текст* другой, используя секретный ключ для *шифрования* (или "засекречивания") сообщения и получая, таким образом, шифртекст, который передается получателю. Получатель применяет тот же ключ для *расшифровки* (или «дешифрования») шифрограммы и восстановления исходного сообщения. Обратите внимание, что один и тот же ключ используется для преобразования открытого текста в шифртекст и наоборот, именно поэтому данный способ известен как *симметричное* шифрование, где под симметрией понимают тот факт, что у сторон имеется один и тот же ключ, который используется для шифрования и расшифрования. Противоположный вариант называется *ассиметричным шифрованием* или *шифрованием* с открытым ключом (рассматривается в Г лаве 10), в котором при шифровании и расшифровке используются разные ключи.

Как уже упоминалось, цель шифрования - скрывать исходный текст от перехватчика, который может отслеживать канал обмена данными и видеть шифрограмму. Далее в этой главе мы более подробно обсудем данную тему , и потра-

тим не мало времени определяя цели в Главах 2 и 3 .

Есть две канонические области применения криптографии с закрытым ключом. В первом случае есть две стороны, разделенные расстоянием, т.е. сотрудница в Нью-Йорке передает сообщение своему коллеге в Калифорнии; см. Рисунок 1.2. Подразумевается, что они заранее до сеанса связи обменялись ключом. (Обратите внимание, что если одна сторона просто посылает ключ другой по открытому каналу связи, то перехватчик получает ключ тоже!) Часто это легко сделать, если стороны организовали очную встречу в безопасном месте для обмена ключом, прежде чем они разъедутся; в предыдущем примере, сотрудники могли встретиться, чтобы обменяться ключом, когда находились в офисе в Нью-Йорке. В других случаях организовать безопасный обмен ключами гораздо сложнее. В последующих главах мы предполагаем, что обмен ключами возможен; мы вернемся к этому вопросу в Главе 10

Второй вариант широкораспространенного применения криптосистемы с закрытым ключом, когда сообщение предназначено одной и той же стороне, но по прошествии какого-то *времени*. (См. Рисунок 1.2.) Рассмотрим, например, шифрование диска, где пользователь зашифровывает какой-то открытый текст и хранит полученный шифртекст на жестком диске; тот же пользователь позднее вернется к ним для расшифровки шифртекста и получения первоначальных данных. Жесткий диск здесь служит в качестве канала передачи данных, который может быть атакован со стороны злоумышленника путем получения доступа к жесткому диску и считывания его содержимого. «Обмен» ключами не представляет теперь сложности, хотя пользователю все еще необходим защищенный и надежный способ запоминания/хранения к люча для использования в будущем.

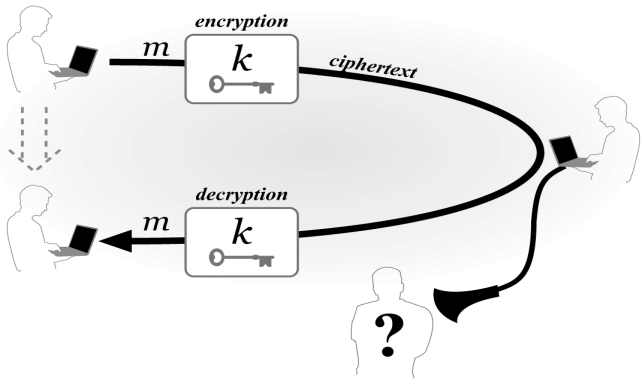

РИСУНОК 1.2: Другой вариант применения криптографии с закрытым ключом (шифрования): пользователь хранит защищенные данные.

Синтаксис шифрования. Строго говоря, схема шифрования с закрытым ключом определяется путем задания *пространства сообщений M* наряду с тремя алгоритмами: генерации ключей *(Gen)*, шифрования *(Enc)* и расшифровки *(Dec)*. Пространство сообщения *M* определяет множество "подлинных" сообщений, т.е ., тех сообщений, которые поддерживаются в данной схеме. Алгоритмы выполняют следующие функции:

1. Алгоритм генерации ключа *Gen* - это вероятностный алгоритм, который выдает ключ *k* , выбираемый согласно некоторой функции распределения.

2. Алгоритм шифрования *Enc* на входе принимает ключ *k* и сообщение m и на выходе выдает шифртекст c. Обозначим через *Enck (m)* шифрование открытого текста m с помощью ключа *k*.

3. Алгоритм расшифрования *Dec* принимает на входе ключ *k* и шифртекст *c* , а на выходе выдает открытый текст *m*. Обозначим расшифрование защищенного текста *c* с помощью ключа *k* через функцию *Dec k(c)*.

Схема шифрования должна удовлетворять следующему требованию корректности: для всякого ключа *k*, формируемого с помощью алгоритма *Gen* и всякого сообщения $m \in M$, справедливо следующее

$$Deck (Enck(m)) = m.$$

Другими словами: шифрование сообщения и затем расшифрование полученного шифртекста (с помощью того же ключа) дает первоначальное сообщение.

Множество всех возможных ключей, генерируемых алгоритмом, называется *пространством ключей* и обозначается *K*. Практически всегда алгоритм *Gen* просто выбирает унифицированный ключ из пространства ключей; фактически, можно предположить без потери обобщения, что это справедливо. (см . упражнение 2.1).

Резюмируя все вышесказанное, можно заключить, что схема шифрования может использоваться двумя сторонами, которые желают обмениваться информацией в следующем порядке. Сначала, запускается алгоритм *Gen* для генерации ключа *k* , которым обмениваются стороны. Затем, когда одной из сторон потребуется отправить исходный текст m другой стороне, она его обработает по *c := Enck (m)* и отправит итоговый шифртекст c по открытому каналу связи принимающей стороне.[1] После получения шифртекста *c*, вторая сторона преобразует его по *m := Deck (c)* для восстановления исходного открытого текста.

Ключи и принцип Керкгоффса. Из вышесказанного вытекает, что если противник знает алгоритм шифрования Dec , а также ключ k, которым обменялись взаимодействующие стороны, то он сможет расшифровать все зашифрованные сообщения, переданные этими сторонами. Именно по этой причине стороны,

осуществляющие обмен данными, должны обмениваться ключами k тайно и хранить ключи k в абсолютной секретности . Может быть им следует хранить в секрете и алгоритм расшифровки Dec ? Исходя из тех же соображений, не лучше было бы им хранить в тайне все сведения о схеме шифрования?

В конце 19 столетия Огюст Керкгоффс утверждал противоположное в своей статье, в которой он разъяснял некоторые принципы разработки криптосистем военного назначения. В одном из наиболее важных пунктов этого письма, ныне известного как *принцип Керкгоффса*, утверждалось следующее:

> *Не нужно, хранить метод шифрования в тайне , а попадание шифра в руки врага не должно создавать проблем.*

То есть, система шифрования должна разрабатываться так, чтобы оставаться надежной, *даже если* противник знает все о системе, до тех пор, пока ему остается неизвестным используемый ключ. Переформулировав, можно сказать, что надежность не должна зависеть от секретности системы шифрования; а наоборот , принцип Керкгоффса требует, чтобы *надежность основывалась исключительно на секретности ключа*.

Существует три основных довода в пользу принципа Керкгоффса. Первый довод состоит в том, что гораздо проще сторонам обеспечить секретность короткого ключа, нежели хранить в секрете весь алгоритм, которым они пользуются . В особенности это справедливо, когда шифрование, допустим, используется для защиты обмена информацией между всеми парами сотрудников в какой-то организации. Если каждая пара не использует свой собственный, уникальный алгоритм, то тогда некоторые пары смогут узнать алгоритм, используемый другими. Утечку информации об алгоритме шифрования может допустить один из сотрудников (после его увольнения, например) либо она может быть получена нападающей стороной с помощью обратного проектирования. В общем предположение о том, что алгоритм шифрования останется секретным, просто нереалистично.

Второй довод состоит в том, что в случае если засекреченная информация, которой обмениваются честные участники, будет все-таки раскрыта, сторонам гораздо легче сменить ключ, чем менять всю систему шифрования. (Сравните о бновление файла с установкой новой программы.) Более того, относительно проще сгенерировать новый секретный ключ, чем организовывать масштабную разработку новой криптосистемы. И, наконец, в случае масштабного

---

[1]Знак ":=" используется для обозначения детерминированного присвоения, и предположим пока, что алгоритм шифрования Enc детерминированный. Перечень принятых обозначений находится в конце книги.

развертывания значительно проще всем пользователям задействовать один и тот же алгоритм/ПО шифрования, чем каждому пользоваться своим собственным алгоритмом. (Справедливо даже для одного пользователя, обменивающегося информацией с несколькими различными сторонами.) По сути, желательно иметь *стандартизированные системы* шифрования, чтобы (1) обеспечивать совместимость по умолчанию и (2) использовать систему, которая испытана в условиях открытого наблюдения, в результате которого не обнаружено никаких слабых мест.

Сегодня принцип Керкгоффса понимают в том смысле , что разработку криптосистем надо вести полностью на открытой платформе, что полностью противоречит понятию «секретность в незаметности» которое предполагает, что хранение алгоритмов в тайне повышает защиту. Весьма опасно использовать «сделанные на коленке» алгоритмы (т.е. не стандартизированные алгоритмы, разрабатываемые в тайне какой-то частной компанией). Наоборот, общедоступные разработки проходят открытый контроль и поэтому вероятно устойчивее. Многолетний опыт показывает, что создание хороших криптографических систем - задача не из легких. Именно поэтому уверенность в защите системы повышается при условии ее всестороннего изучения (со стороны экспертного сообщества, а не разработчиков криптосистемы), в результате которого подтверждается отсутствие уязвимостей. Несмотря на всю простоту и очевидность, принцип открытого шифрования (т. е. принцип Керкгоффса) неоднократно игнорировался с катастрофическими результатами. К счастью, сегодня достаточно надежных, стандартных и широко применяемых криптосистем, что просто незачем использовать что-то еще.

1.3 Исторические шифры и их криптоанализ

В рамках изучения «классической» криптографии мы исследуем некоторые исторические системы шифрования и укажем на их ненадежность. Ознакомление с данным материалом служит нескольким основным целям: (1) указание на недостатки «специализированного» подхода к криптографии, тем самым побуждая на применение современного, строгого подхода , который рассматривается в остальной части книги, и (2) демонстрация того, что простые подходы вряд ли позволят достичь криптостойких систем шифрования. П опутно будут представлены центральные положения криптографии, появившиеся благодаря недостаткам исторических систем.

В данном разделе знаки открытого текста напечатаны нижним регистром, а знаки шифртекста написаны ВЕРХНИМ для ясности и удобства оформления.

Шифр Цезаря . Один из самых старейших зафиксированных шифров, известный как *Шифр Цезаря*, описан в *De Vita Caesarum, Divus Iulius* («Жизнеописание Цезарей, Поклонение Юлию»), составленном приблизительно в 110 г. н.э.:

Также есть его письма к Цицерону и своим близким, о личных делах. Если он хотел передать что-то секретное, то он шифровал письмо, то есть, он менял порядок букв алфавита, чтобы ни слова невозможно было разобрать .

Юлий Цезарь зашифровывал свои письма путем сдвига букв алфавита на три позиции вперед: буква «a» заменялась «D», буква «b» на E и так далее . В конце алфавита происходил циклический переход, и поэтому буква «z» заменялась на «C», буква «y» на «B», а буква «x» на «A». Например, в результате шифрования сообщения *begin the attack now*, без пробелов, получалось:

EHJLQWKHDWWDFNQRZ.

Непосредственная проблема данного шифра в том, что метод шифрования *фиксированный;* ключ отсутствует вовсе. Таким образом, любой, кто мог узнать как Цезарь шифровал свои сообщения, был способен без труда дешифровать сообщения.

Интересно, что вариант такого шифра под названием ROT-13 (в котором сдвиг составляет 13 знаков вместо 3) до сих пор используется в различных сетевых форумах. Понятно , что такой метод не дает никакой криптозащиты; он используется просто для того, чтобы текст (например, спойлер-анонс киноновинки) не читался, если только сам пользователь не решал намеренно его расшифровать.

Сдвиговый шифр и принцип достаточного пространства ключей. *Сдвиговый шифр* можно рассматривать, как вариант шифра Цезаря[2] только с ключом. В частности, в сдвиговом шифре ключ *k является* номером между 0 и 25. Чтобы зашифровать, буквы сдвигаются как в шифре Цезаря, но количество знаков определяется ключом k . Переводя это на синтаксис шифрования, описанный выше, получаем, что пространство сообщений состоит из цепочек букв латинского алфавита произвольной длины без использования знаков пунктуации, пробелов и числительных, а также нечувствительных к регистру. Алгоритм генерации *Gen* выдает унифицированный ключ $k \in \{0, \ldots, 25\}$; алгоритм *Enc* принимает ключ k и открытый текст, сдвигает каждую букву открытого текста вправо на k мест (циклический переход в конце алфавита); а алгоритм расшифровки *Dec* принимает ключ k и шифртекст и сдвгает каждую букву сообщения влево на k мест.

Получается строго математическое описание , если приравнять английский алфавит с множеством $\{0, \ldots, 25\}$ (так,a = 0, b = 1, и т.д .).

---

[2]В некоторых книгах, “шифр Цезаря” и “сдвиговый шифр” используются взаимозаменяемо.

Тогда пространство сообщений M - любая конечная последовательность целых чисел из этого множества. Шифрование сообщения m = m1 • • • mA (где mi $\in \{0, \ldots, 25\}$) с помощью ключа k задается выражением

$Enck\,(m1 \bullet\bullet\bullet mA)$ = c1 • • • cA, где ci = [(mi + k) mod 26].

(Запись $[a \bmod N]$ обозначает остаток от деления на N, при $0 \le [a \bmod N] < N$. Мы называем процесс отображения a в $[a \bmod N]$ как деление по модулю N; мы поговорим об этом подробнее в начале Г лавы 8.) Расшифровка шифртекста c = c1 • • • cA с помощью ключа k задана выражением

Dec$k$ (c1 • • • cA) = m1 • • • mA, где $mi = [(ci - k) \bmod 26]$.

Хорошо ли защищен сдвиговый шифр? Перед тем как продолжить чтение, попытайтесь расшифровать следующий текст, зашифрованный с помощью сдвигового шифра и секретного ключа k:

OVDTHUFWVZZPISLRLFZHYLAOLYL.

Можно ли восстановить сообщение не зная k? На самом деле все просто! Причина в том, что всего существует только 26 возможных ключей. Поэтому можно попытаться расшифровать шифрограмму с помощью перебора всех возможных ключей и тем самым получить 26 образцов открытого текста. Несомненно среди них будет и верный образец; более того, если шифртекст «достаточно большой», тогда правильный текст будет по всей вероятности единственным образцом среди всех, который «обладает смыслом». (Последнее утверждение необязательно должно быть верным, но оно справедливо в большинстве случаев. Даже когда это не так, атака сужает множество потенциальных текстов максимум до 26.) Просмотрев весь перечень из 26 образцов, легко обнаружить первоначальный открытый текст.

Атака, которая заключается в переборе всех возможных ключей, называется *атакой методом грубой силы* или *перебором всех возможных вариантов*. Очевидно, что стойкая криптосистема не должна быть уязвима атакам такого рода.[3] Данное замечание известно, как п*ринцип достаточности пространства ключей:*

> *В любой стойкой криптосистеме пространство ключей должно быть достаточно большим, чтобы свести на нет любые попытки атак методом перебора.*

Можно дискутировать на тему того, сколько попыток должно быть, чтобы считать взлом «неосуществимым», а точное определение осуществимости зависит, как от ресурсов потенциальной нападающей стороны, так и от длительности времени, в течение которого отправляющая и принимающая стороны хотят обеспечивать секретность обмена данными. Сегодня злоумышленники могут использовать суперкомпьютеры, тысячи персональных компьютеров или

графических процессоров для ускорения атак методом грубой силы. Для защиты от таких атак необходимо, чтобы пространство ключей было очень большим - скажем, размером как минимум 270 и даже больше, если имеется заинтересованность в длительной защите от нападающих противников с хорошим финансированием.

Принцип достаточности пространства ключей предоставляет *необходимое* условие для безопасности, но НЕ достаточное. Это проиллюстрировано в следующем примере.

Моноалфавитные подстановочные шифры . В подстановочном шифре ключ определяет отображение каждой буквы алфавита (открытого текста) в какой-то букве алфавита (шифртекста), где отображение является фиксированным сдвигом, которое определяется ключом. В случае *моноалфавитного подстановочного шифра*, ключ также устанавливает отображение элементов из пространства исходных сообщений элементами из пространства зашифрованных, при этом такое отображение может иметь *произвольный характер* с единственным ограничением в том, что оно осуществляется один к одному, а поэтому вероятность дешифрования сохраняется. Ключевое пространство, таким образом, состоит из всех *взаимно-однозначных соответствий* или *перестановок* элементов алфавита. Например, ключ, который определяет следующую перестановку,

a b c d e f g h i j k l m n o p q r t v w x y z
X E U A D N B K V M R O C Q F S Y H W G L Z I T

(в которой буква «а» отображается в букву «X» и т.д..) преобразует сообщение tellhimaboutme в GDOOKVCXEFLGCD. Название данного шифра происходит из того факта, что ключ определяет (фиксированную) замену отдельных символов открытого текста.

Полагая, что используется английский алфавит, пространство ключей имеет размер 26! = 26 • 25 • 24 • • • 2 • 1 или приблизительно 288 и, следовательно, атака методом грубой силы неосуществима. Это, тем не менее, не означает, что шифр надежный . На самом деле, как мы покажем дальше, данную криптосистему легко вскрыть даже несмотря на большое пространство ключей.

Предположим, что шифруется текст на английском языке (т.е. текст, написанный и оформленный по правилам английской грамматики, а не просто какой-то набор букв английского алфавита). В таком случае атакующая сторона может

[3]Формально, это справедливо только тогда, если пространство сообщений больше, чем пространство ключей; это будет рассматриваться в главе 2. Это характерная особенность для криптосистем, применяемых на практике.

попытаться вскрыть текст, зашифрованный с помощью моноалфавитного подстановочного шифра, прибегнув к анализу языкового узора английского языка. (Конечно же, такие действия сработают для любого другого языка). Подобная атака основана на следующих фактах:

1. Независимо от ключа, отображение каждой буквы фиксировано, и поэтому если буква «e» преобразовано в букву «D», то всякий раз, когда встречается буква «e» в открытом тексте, в шифртексте будет встречаться буква «D» .

2. Известно распределение частотности отдельных букв в английском языке (см. Р исунок 1.3). Естественно, в очень коротких текстах может встречаться отступление от этого правила, но даже в текстах, состоящих из нескольких предложений, частотность появления близка к средней.

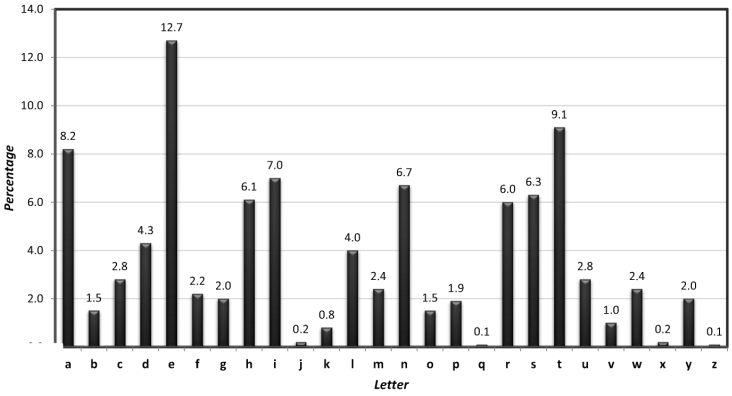

РИСУНОК 1.3: Средняя частота повторения букв в текстах на английском языке.

Атака основана на классифицировании распределений частот ности символов в шифртексте, т.е., на ведении записей, что буква «A» встретилась 11 раз, а буква «B» встретилась 4 раза и так далее. Данные частоты затем сравниваются с известными частотами повторения букв в обычном тексте на английском языке. На основе замеченных закономерностей можно разгадать остальные части отображения, заданного ключом. Например, поскольку буква «e» наиболее часто встречаемая буква англйского языка, можно догадаться, что самый часто встречаемый знак в шифртексте соответствует букве «e» открытого текста и так далее. Некоторые из предположений могут оказаться неверными, но как только число правильных догадок достигнет достаточного, дешифрование займет относительно немного времени (особенно при использовании и других сведений об английском языке, например то, что «u» обычно следует за «q» ,

а также то, что «h» с наибольшей вероятностью встретится между «t» и «e»). Можно заключить, что хотя моноалфавитный подстановочный шифр обладает огромным пространством ключей, он не является криптостойким.

Не стоит удивляться тому, что моноалфавитный подстановочный шифр можно быстро вскрыть, поскольку в газетах то и дело печатают головоломки на основе данного шифра (которые некоторые решают еще до утреннего кофе!). Рекомендуем вам попробовать дешифровать следующий шифртекст - В ы убедитесь как легко выполнить атаку на данный шифр. (Пользуйтесь Р исунком 1.3 для справки.)

JGRMQOYGHMVBJWRWQFPWHGFFDQGFPFZRKBEEBJIZQQOCIBZKLF
AFGQVFZFWWE OGWOPFGFHWOLPHLRLOLFDMFGQWBLWBWQOLK
FWBYLBLYLFSFLJGRMQBOLWJVFP FWQVHQWFFPQOQVFPQOCFPOG
FWFJIGFQVHLHLROQVFGWJVFPFOLFHGQVQVFILE OGQILHQFQGIQV
VOSFAFGBWQVHQWIJVWJVFPFWHGFIWIHZZRQGBABHZQOCGFHX

Усовершенствованная атака на сдвиговый шифр. Можно использовать таблицы частотности букв, чтобы провести более изощренную атаку на сдвиговый шифр. В предыдущих атаках на сдвиговый шифр требовалось дешифровать засекреченный текст с помощью каждого возможного ключа и затем проверять какой ключ дает образец «со смыслом». Недостаток данного подхода в том, что его до некоторой степени проблематично автоматизировать, поскольку машине трудно определить, имеет ли данный образец открытого текста «смысл» или нет. (Мы не утверждаем, что это невозможно, так как криптоанализ можно автоматизировать с помощью современного английского толкового словаря. Мы лишь утверждаем, что данная задача автоматизации отнюдь не тривиальна.) Кроме того, могут быть случаи - мы рассмотрим один такой позднее - когда буквы открытого текста распределены в соответствии с характеристиками частотности английского текста, хотя сам текст составлен не на общепринятом английском языке, в результате проверка образца «на смысл» не сработает.

Опишем теперь вариант атаки, который не имеет данных недостатков. Как и до этого, интерпретируем буквы английского алфавита с цифрами: 0, . . . , 25. Пусть *pi*, при условии $0 \le pi \le 1$, обозначает частоту встречаемости i-той буквы в обычном тексте на английском (без учета пробелов, знаков пунктуации и т.д.). Вычисление по схему на рисунке 1.3. даст

$$\sum_{i=0}^{25} p_i^2 \approx 0.065. \qquad (1.1)$$

Теперь, допустим, мы получили некоторый шифртекст, и пусть *qi* обозначает частоту *i*-той буквы алфавита в данном шифртексте, т.е. *qi* - просто число появлений *i*-той буквы алфавита в шифртексте, поделенное на длину зашифрован-

ного текста. Если ключ - *k*, следовательно, pi должно быть примерно равно *qi*+*k* для всех *i*, так как i-тая буква отображается в (i + *k*)-тую букву. (Мы используем i+k вместо более громоздкой структуры [*i*+*k* mod 26].) Таким образом, если вычислить

$$I_j \stackrel{\text{def}}{=} \sum_{i=0}^{25} p_i \cdot q_{i+j}$$

для каждого значения $j \in \{0, \ldots, 25\}$, то следует ожидать, что $Ik \approx 0{,}065$ (где *k* - настоящий ключ), тогда как *Ij* для *j f* = *k* будет отлично от 0,065. Это приводит к атаке на восстановление ключа, которую легко автоматизировать: вычислить Ij для всех j, а затем вывести значение k для которого Ik ближе всего к 0,065.

Шифр Виженера (полиалфавитные сдвиговые). Статистическая атака на моноалфавитный подстановочный шифр возможна по причине того, что ключ определяет фиксированное отображение, которое применяется побуквенно к открытому тексту. Расстроить подобную атаку можно с помощью *полиалфавитного подстановочного шифра*, в котором ключ определяет отображение, которое применяется к блокам букв шифруемого текста. Например, ключ может заменить двухбуквенный блок «ab» на «DZ» , а «ac» отобразить через «TY» ; заметьте, что буква «a» открытого текста не заменяется фиксированным элементом шифртекста. Полиалфавитные подстановочные шифры «размывают» частотные характеристики распределения букв в шифртекстах, тем самым затрудняя проведение статистического анализа.

Шифр Виженера, особый случай выше описанного, также называемый полиалфавитным *сдвиговым* шифром, работает путем применения последовательности нескольких отдельных образцов сдвигового шифра. Ключ теперь рассматривается как *строка букв*; шифрование выполняется путем сдвига каждой буквы открытого текста на число позиций, указанное следующим элементом ключа, записанного циклически до тех пор, пока его длина не будет соответствовать длине исходного текста. (Если ключ имеет длину 1, то шифр становится простым сдвиговым) Например,

сообщение tellhimaboutme шифруется с помощью ключа cafe

следующим образом:

Открытый текст:

tellhimaboutme

Ключ (повторяющийся):

cafecafecafeca

Шифртекст:

VEQPJIREDOZXOE

(Необязательно, чтобы ключ был английским словом.) Точно так же как шифрование первого, пятого, девятого, . . . символа с помощью сдвигового шифра и ключа c; второго, шестого, десятого, . . . символа - ключом a; третьего, седьмого, . . . символа - ключом f; а четвертого, седьмого . . . символа - ключом e. Стоит отметить, что в приведенном выше примере буква l преобразована один раз в Q и один раз в P. К тому же, зашифрованный текст содержит символ E , полученный в одном случае из буквые , а в другом - из буквы a. Таким образом, характеристики частот появления символов в шифртексте «размыты», что и требовалось.

Если ключ достаточно длинный, то его взлом представляется далеко не простой задачей. Более того, многие считали его «невзламываемым», и хотя его изобрели в 16 веке, методика его взлома появилась лишь сотни лет спустя.

Атака на шифр Виженера. Первое, на что надо обратить внимание при попытке раскрыть шифр Виженера, - это *известна ли длина ключа*, остальной анализ шифра относительно прост. В частности, пусть длина ключа, также называемая *периодом повторения*, равна t. Запишем ключ k как $k = k1 \bullet\bullet\bullet kt$, где каждый ключ ki - буква алфавита.

Наблюдаемый шифркод $c = c1c2 \bullet\bullet\bullet$ можно поделить на t частей, где каждая часть может рассматриваться как шифрграмма с помощью сдвигового шифра. В частности, для всех $j \in \{1, \ldots, t\}$ символы шифртекста

$$cj,\ cj+t,\ cj+2t, \ldots$$

получились путем сдвига соответствующих букв открытого текста на kj позиций. Последовательность символов, которая указана выше, называется j-тым *потоком*. Все, что осается сделать - определить какой из 26 возможных сдвигов использовался для каждого из потоков t . Данная задача не так проста, как в случае со сдвиговым шифром, так как больше уже невозможно простым перебором определить, когда дешифрование потока будет «иметь смысл». (Стоит напомнить, что поток не соответствует последовательности букв открытого текста.) Далее, в попытке разгадать весь ключ k сразу потребуется перебрать 26^t различных вариантов, что нереализуемо при больших t. Несмотря на это, можно применять частотный анализ для вскрытия каждого потока отдельно. То есть для каждого потока мы составляем таблицу частотности каждого символа шифртекста и проверяем, какой из 26 возможных сдвигов приводит к «правильному» распределению вероятностей в данном потоке. Поскольку такая процедура может проводиться для каждого потока в отдельности (т.е. для каждого символа ключа), времени на такую атаку потребуется $26 \bullet t$, а не 26^t.

Более принципиальный и легкий для автоматизации подход заключается в ис-

пользовании улучшенного метода раскрытия сдвигового шифра, описанного ранее. Такая атака не опиралась на проверку образца «на смысл», а только учитывала базовое частотное распределение символов открытого текста.

Любой из представленных выше подходов подойдет для атаки, когда известна длина ключа. А если она неизвестна?

Поскольку максимальная длина ключа T не слишком велика, мы можем запросто повторить описанную выше атаку T раз (для каждого возможного значения $t \in \{1, ..., T\}$). В результате мы получим максимум T разных потенциально подходящих текстов, среди которых выявить настоящий текст, скорее всего, будет просто. Таким образом, неизвестная длина ключа не является серьезным препятствием.

Кроме того, существуют более эффективные способы определения длины ключа по наблюдаемому шифртексту. Один из них - *метод Касиски*, опубликованный в середине XIX века. Первый шаг - выявление повторяющихся сочетаний двух или трех символов шифртекста. Они, скорее всего, будут соответствовать часто встречающимся биграммам или триграммам открытого текста. Например, рассмотрим часто встречающееся в английском языке слово «the». Это слово будет отображаться с помощью разных символов шифртекста, в зависимости от его положения в открытом тексте. Однако если слово дважды встретится в относительно похожей позиции, то оно отобразится с помощью аналогичных символов шифртекста. Таким образом, в достаточно длинном открытом тексте, велика вероятность, что «the» неоднократно отобразится с помощью одинаковых символов шифртекста.

Рассмотрим конкретный пример, ключом в котором будет beads (пробелы добавлены для внесения ясности):

Открытый текст:

the man and the woman retrieved the letter from the post office (мужчина и женщина получили письмо на почте) Ключ: bea dsb ead sbe adsbe adsbeadsb ead sbeads bead sbe adsb eadsbe Шифртекст: ULE PSO ENG LII WREBR RHLSMEYWE XHH DFXTHJ GVOP LII PRKU SFIADI

Слово «the» иногда отображается как ULE, иногда как LII, и иногда как XHH. С другой стороны, оно *дважды* встречается в виде LII, а в достаточно длинном тексте оно, скорее всего, несколько раз встретилось бы в каждом из возможных сочетаний. Наблюдение Касиски заключалось в том, что расстояние между подобными повторениями (при условии, что они неслучайны) должно быть кратно периоду повторения. (В приведенном выше примере период повторения

равен 5, а расстояние между двумя сочетаниями LII равно 30, что соответствует 6 повторениям периода.) Таким образом, наибольший общий делитель расстояния между повторяющимися последовательностями (при условии, что они неслучайны) позволяет определить длину ключа t или его кратное.

Альтернативный подход, *метод индекса повторений*, методичнее, а значит, его легче автоматизировать. Напомним, что если длина ключа равна t, то все символы шифртекста

$$c1,\ c1+t,\ c1+2t,\ \ldots$$

в первом потоке были получены из шифрования с использованием одинакового сдвига. Таким образом, предполагается, что частота появлений символов в данной последовательности будет совпадать с частотой появлений символов стандартного английского текста *в некоторой сдвинутой последовательности*. Например, пусть *qi* обозначает замеченную частоту повторений *i*-той буквы английского алфавита в данном потоке, соответствующее отношению числа вхождений *i*-той буквы алфавита к общему числу букв в потоке.

Если использованный здесь сдвиг равен *j* (т.е ., если первый символ ключа *k1* равен *j*), то для всех *i* мы полагаем, что $qi+j \approx pi$, где *pi* - частота появления *i*-той буквы алфавита стандартного английского текста. (Опять-таки , мы используем $qi+j$ вместо $q[i+j \bmod 26]$.) Но из этого следует, что последовательность *q0, ..., q25* всего лишь является последовательностью *p0, ..., p25*, сдвинутой на *j* мест. Следовательно, (см. Уравнение (1.1)):

$$\sum_{i=0}^{25} q_i^2 \approx \sum_{i=0}^{25} p_i^2 \approx 0.065.$$

Таким образом, мы можем точно определить длину ключа *t*. Для $\tau = 1, 2, \ldots$, рассмотрите последовательность символов шифртекста c1, $c1+\tau$, $c1+2\tau$, ... и составьте таблицу $q0, \ldots, q25$ для данной последовательности. Затем вычислите

$$S_\tau \stackrel{\text{def}}{=} \sum_{i=0}^{25} q_i^2$$

Когда $\tau = t$, мы предполагаем, что $S\tau \approx 0.065$, как указано выше. С другой стороны, если τ не кратно *t*, мы предполагаем, что все символы будут встречаться с приблизительно равной вероятностью в последовательности $c1, c1+\tau, c1+2\tau, \ldots$, а значит, мы полагаем, что $qi \approx 1/26$ для всех *i*. В этом случае мы получим

$$S_\tau \approx \sum_{i=0}^{25} \left(\frac{1}{26}\right)^2 \approx 0.038.$$

Таким образом, наименьшее значение τ, для которого $S\tau \approx 0.065$, скорее всего, будет равно длине ключа. Затем можно проверить предположение τ, проведя

аналогичные расчеты, используя второй поток $c2$, $c2+\tau$, $c2+2\tau$, . .., и т.д .

Длина шифртекста и криптоаналитические атаки. Вышеописанные атаки на шифр Виженера требуют более длинного шифртекста, чем атаки на предшествующие системы. Например, для работы с методом индекса повторений необходимо, чтобы последовательность *c1, c1+t, c1+2t* (где *t* - действительная длина ключа) была достаточно длинной для обеспечения совпадения замеченных повторений с предполагаемыми. В таком случае сам шифртекст должен быть примерно в *t* раз больше. Аналогичным образом, описанная нами атака на моноалфавитный шифр подстановки требует шифртекста большего размера, чем для атаки на шифр сдвига (которая может оказаться успешной при шифровании даже одного слова). Таким образом, мы видим, что чем длиннее ключ, тем, как правило, больше шифртекст требуется криптоаналитику для проведения атаки. (Действительно , шифр Виженера может оказаться надежным, если ключ имеет ту же длину, что и шифруемый текст. Аналогичный феномен мы рассмотрим в следующей главе.)

Выводы. Мы представили лишь немногие исторические шифры. Помимо исторического экскурса нашей целью было продемонстрировать с их помощью несколько важных уроков. Возможно, самое важное в них то, что *создать надежный шифр сложно*. Шифр Виженера долгое время сохранял свою надежность. Кроме того, применялись гораздо более сложные системы. Однако сложная система не всегда надежна, и все исторические системы были взломаны.

1.4 Принципы современной криптографии

Как, должно быть, уже стало ясно из предыдущего раздела, в прошлом криптография в большей степени была искусством, а не наукой. Системы разрабатывались нерегулярно и оценивались с точки зрения их сложности и оригинальности. Если бы систему оценивали с точки зрения ее уязвимости для атак, то ее приходилось бы постоянно «латать», чтобы предотвратить атаку, а затем повторять процесс. И хотя вполне могла существовать договоренность о ненадежности некоторых систем (о чем свидетельствуют особенно вредоносные атаки), общепринятые требования, предъявляемые к «надежной» системе, а также доказательства того, что какая-то конкретная система является таковой, отсутствовали.

За несколько последних десятилетий криптография превратилась в науку. Сейчас разработка и анализ систем проводятся более методично.

Конечной целью является предоставление строгого *доказательства* того, что данная конструкция безопасна. Чтобы сформулировать такие доказательства, нам сначала необходимо обратиться к *формальным определениям* «надежный», которые полезны и интересны сами по себе. Оказалось, что большая часть криптографических доказательств основывается на недоказанных в настоящий

момент гипотезах об алгоритмической неразрешимости определенных математических задач. Любая такая гипотеза должна быть явно выражена и точно сформулирована. Именно упор на определения, допущения и доказательства отличает современную криптографию от классической. В следующих разделах мы подробно обсудим эти три принципа.

1.4.1 Принцип 1 – Формальные определения

Одним из ключевых достижений современной криптографии стало осознание того, что формальные определения безопасности имеют важное значение для правильного проектирования, исследования, оценки и использования криптографических примитивов. Иными словами:

> *Если вы не знаете, чего хотите достичь, как вы определите, когда вы этого достигли (и достигли ли)?*

Формальные определения обеспечивают такое понимание, описывая охватываемые угрозы и желаемые гарантии безопасности. Таким образом, определения могут направлять разработку криптографических систем. Действительно, лучше формализовать требования до начала работы над проектом, чем *потом* подгонять определение, когда процесс уже завершен. Последний подход чреват тем, что этап проектирования закончится вместе с терпением разработчика (а не после достижении цели), или же конструкция достигнет *большего*, чем требовалось, ценой эффективности.

Определения также позволяют оценить и проанализировать полученную конструкцию. С помощью имеющегося определения можно изучить предлагаемую систему и выявить, обеспечивает ли она желаемые гарантии, а в некоторых случаях даже доказать ее безопасность (см. раздел 1.4.3), подтвердив ее соответствие определению. В то же время, опираясь на определения, можно достоверно доказать, что данная система небезопасна, поскольку не соответствует определению. В частности, обратите внимание на то, что описанные в предыдущем разделе атаки не доказывают по умолчанию «ненадежность» описанных систем. Например, атака на шифр Виженера предполагает шифрование достаточно длинного текста на английском языке. Но будет ли шифр Виженера «надежным» для короткого или сжатого текста на английском языке, в котором едва ли встретятся стандартные повторения букв? Сложно ответить на этот вопрос, не имея формального определения.

Определения позволяют корректно сравнивать системы. Как мы вскоре увидим, существует множество приемлемых определений безопасности, однако, «правильное» зависит от условий, в которых будет применяться система. Система, соответствующая более слабому определению, может оказаться эффективнее системы, соответствующей более сильному определению. С помощью

точных определений мы можем правильно оценить различия между системами. Аналогичным образом определения обеспечивают безопасное использование систем. Рассмотрим принятие решения о том, какая система шифрования подойдет для приложения большего размера. Надежным решением будет сначала максимально точно определить понятие безопасности в контексте данного приложения, а затем найти соответствующую ему систему шифрования. Дополнительным преимуществом такого подхода является *модульность*: разработчик может заменить одну систему шифрования на другую, также соответствующую необходимому определению безопасности, и не волноваться по поводу безопасности приложения в целом.

Написание формального определения заставляет задуматься о том, что существенно для рассматриваемой задачи и какие параметры с ней не связаны. Зачастую во время работы выявляются такие тонкости, которые не были заметны в начале. Давайте рассмотрим это на примере шифрования.

Пример: надежное шифрование. В этом случае распространенной ошибкой будет пренебречь необходимостью формального определения или же использовать поверхностное определение, потому что «каждый интуитивно догадывается о том, что такое надежность». Но это не так. Рассмотрим в качестве примера случай шифрования. (Здесь читатель, вероятно, захочет подумать над тем, как бы он формально определил надежную систему шифрования.) И хотя мы отложим формальное определение надежного шифрования до двух следующих глав, здесь мы попробуем описать, что такое определение должно в себя включать.

Как правило, понятие надежности состоит из двух компонентов: гарантии безопасности (или, с точки зрения перехватчика, что определяет успех атаки на данную систему) и модели угроз. Гарантия безопасности определяет действия злоумышленника, на предотвращение которых направлена система. Модель угроз описывает силу противника, т.е. гипотезы о том, какие действия может предпринять злоумышленник.

Давайте начнем с первой составляющей. Какие гарантии должна предоставлять система надежного шифрования? Вот некоторые мысли на этот счет:

• *Перехватчик не должен суметь восстановить ключ.* Ранее мы уже говорили о том, что если перехватчик может определить ключ, используемый двумя сторонами, применяемая схема не является надежной. Однако достаточно просто представить систему, восстановление ключа для которой будет невозможно, а сама система будет абсолютно ненадежной. Рассмотрим, например, систему, где $Enc_k(m) = m$. Шифртекст не содержит информации о ключе (и ключ не может быть восстановлен, если текст достаточно длинный), но направляется-то незашифрованное сообщение! Таким образом, мы видим, что невозможности

восстановить ключ недостаточно для надежности системы. Смысл в том, что цель шифрования - защитить *сообщение*, а ключ - лишь средство для достижения цели и сам по себе не важен.

• *Перехватчик не должен суметь восстановить весь открытый текст из шифртекста.* Такое определение лучше, но еще далеко от удовлетворительного. В частности, такое определение подразумевает, что система шифрования считается надежной, если из шифртекста было получено 90% открытого текста, а оставшиеся 10% расшифровать не удалось. Очевидно, что такой подход неприемлем для большинства случаев использования шифрования. Например, при шифровании базы данных заработной платы: мы будем небезосновательно встревожены, когда 90% данных о заработной плате сотрудников будут рассекречены!

• *Перехватчик не должен суметь восстановить из шифртекста ни одного символа открытого текста.* Такое определение кажется хорошим, но недостаточно. Вернемся к примеру с шифрованием базы данных заработной платы, мы не можем считать систему шифрования надежной, если даже не видя конкретных цифр, можно определить выше или ниже 100 000 долларов США заработная плата сотрудника. Аналогичным образом, нежелательно, чтобы система шифрования позволяла определить, зарабатывает ли сотрудник А больше сотрудника В.

Другой вопрос состоит в том, как формализовать, что для противника значит «восстановить символ шифртекста». А что если перехватчик правильно, совершенно случайно или благодаря сторонней информации, угадает, что цифра младшего разряда - это 0? Несомненно, что этим нельзя оправдывать ненадежность системы шифрования, а значит любое подходящее определение должно исключать такое развитие событий как успешную атаку.

• *«Правильный» ответ: независимо от того, какой информацией уже обладает перехватчик, шифртекст не должен содержать дополнительной информации об открытом тексте, лежащем в его основе.* Неформальное определение охватывает все описанные выше проблемы. В частности, обратите внимание на то, что в нем не определяется, какая информация об открытом тексте является «значимой», в нем просто говорится, что никакая информация не должна быть раскрыта. Это важно, поскольку означает, что система надежного шифрования подходит для всех возможных применений, требующих секретности.

Чего здесь не хватает, так это точного, математического описания определения. Как отразить первоначальную информацию, которой располагает перехватчик об открытом тексте? Что значит (не) допустить утечку информации? Мы вернемся к этим вопросам в двух следующих главах, особенное внимание уделите определениям 2.3 и 3.12.

Теперь, когда мы определили *цель* безопасности, осталось задать *модель угрозы.* Она определяет, какой «силой» предположительно обладает перехватчик, но ничем не ограничивает *стратегию противника*. В этом и заключается важное отличие: мы задаем предполагаемые возможности злоумышленника, но не предполагаем, *как он их использует*. Невозможно предвидеть, какие стратегии могут использоваться в ходе атаки, и, как показывает история, такие попытки обычно обречены на провал.

Существует несколько вероятных моделей угроз в контексте шифрования. Ниже перечислены стандартные модели в порядке возрастания силы перехватчика:

• **Атака на основе шифртекста**: Это самый основной вид атаки, при которой перехватчик только анализирует шифртекст (или несколько шифртекстов) и пытается определить информацию, относящуюся к открытому тексту (или открытым текстам). Такую модель угрозы мы косвенно допускали в ходе обсуждения классических систем шифрования в предыдущем разделе.

• **Атака на основе открытых текстов:** В этом случае перехватчик в состоянии изучить одну или несколько пар «открытый текст - шифртекст», созданных с помощью некоторого ключа. Цель перехватчика - получить информацию об открытом тексте какого-либо другого шифртекста, созданного с использованием того же ключа.

Все рассмотренные нами классические системы шифрования обычно взламываются с использованием атаки на основе открытых текстов. Пример будет представлен в виде упражнения.

• А**така на основе подобранного открытого текста :** В ходе такой атаки перехватчик может получить пары «открытый текст - шифртекст» (как описано выше) для открытых текстов *по своему усмотрению*.

• **Атака на основе подобранного шифртекста:** Последний тип атаки характеризуется тем, что перехватчик может *расшифровать* (получить некоторую информацию о расшифровке) шифртекста на выбор, например, является ли расшифровкой некоторого подобранного перехватчиком шифртекста имеющее значение сообщение на английском языке. Повторимся, что целью перехватчика является получение информации об открытом тексте какого-либо другого шифртекста, чью расшифровку он не может получить напрямую.

Ни одна из этих моделей угроз не является по своей природе лучше любой другой, поскольку правильное использование, зависит от среды, в которой развернута система шифрования.

Первые два типа атак - самые простые в реализации. Для атаки на основе шифртекста злоумышленнику требуется только перехватить по каналу общедоступной связи отправленные зашифрованные сообщения. При атаке на основе

открытых текстов предполагается, что перехватчик каким-то образом смог также получить шифртексты, относящиеся к известным открытым текстам. Часто это легко осуществить, поскольку не все зашифрованные сообщения являются конфиденциальными, по крайней мере, не бесконечно. Простой пример: две стороны могут всегда шифровать приветственное сообщение “hello” в начале каждого установления связи. Более сложный пример: для сохранения в тайне данных ежеквартального отчета о прибыли до даты его обнародования применяется шифрование, тогда в случае перехвата шифртекста позже можно получить соответствующий открытый текст.

В двух последних типах атак предполагается, что перехватчик получил зашифрованные и (или) расшифрованные открытые тексты и шифртексты по своему усмотрению. Сначала это может показаться странным, но мы подробно обсудим эти виды атак и их практическую применимость в разделах 3.4.2 (атака на основе подобранного открытого текста) и 3.7 (атака на основе подобранного шифртекста).

1.4.2 Принцип 2 – Точные гипотезы

Безусловная надежность большинства современных криптографических конструкций не может быть доказана, поскольку такие доказательства требовали бы решения вопросов вычислительной сложности, которые сегодня остаются нерешенными. Результатом такого досадного положения вещей является то, что доказательства стойкости обычно основываются на *гипотезах*. Современная криптография требует того, чтобы каждая такая гипотеза была явно выраженной и математически точной. На самом базовом уровне это обусловлено требованиями к математическим доказательствам надежности . Однако существуют и другие причины:

1. *Подтверждение гипотез:* По своей природе гипотезы - это утверждения, которые не были доказаны, но считаются истинными. Для укрепления нашей уверенности в истинности какой-либо гипотезы, ее необходимо изучить. Наша уверенность в истинности гипотезы зависит от того, как хорошо она изучена и проверена, была ли опровергнута. К тому же изучение гипотезы может подтвердить ее, предоставив доказательства того, что другая широко распространенная гипотеза подразумевает ее истинность.

Если гипотеза, на которую мы опираемся, не была точно сформулирована, ее нельзя изучить и, вероятно, опровергнуть. Таким образом, непременным условием увеличения нашей уверенности в гипотезе является точная формулировка того, что именно мы допускаем.

2. *Сравнение систем*: Часто в криптографии нам представляют две системы, доказательство соответствия некоторому определению которых основывается на разных гипотезах. Какой системе при прочих равных отдать предпочтение? Если первая гипотеза, лежащая в основе первой системы, *слабее* гипотезы, ле-

жащей в основе второй системы (т.е. вторая гипотеза подразумевает первую), тогда предпочтение отдается первой системе, поскольку может оказаться, что вторая гипотеза неверна, хотя верна первая. Если гипотезы, лежащие в основе двух систем не сопоставимы, тогда, как правило, предпочтение отдается системе, основанной на более изученной и надежной гипотезе.

3. *Понимание необходимых гипотез:* Система шифрования может основываться на некотором базовом структурном элементе. Как мы определим, сохраняет ли система шифрования свою надежность, если позже будут обнаружены слабые места структурного элемента? Если гипотезы, касающиеся структурного элемента выявлены во время доказательства надежности системы, тогда нам достаточно проверить, повлияли ли обнаруженные слабые места на необходимые допущения.

Вопрос, который иногда возникает: почему бы не допустить, что конструкция надежна *сама по себе,* вместо того, чтобы доказывать надежность системы, опираясь на другую гипотезу? В некоторых случаях такой подход разумен, например, когда система на протяжении многих лет успешно противостоит атакам. Но такой подход совершенно нежелателен и исключительно опасен при внедрении новой системы. Указанные выше причины объясняют, почему. Во-первых, гипотеза, проверяемая на протяжении нескольких лет, предпочтительнее нового, предположения, специально сделанного и введенного вместе с новой конструкцией. Во-вторых, обычно предпочтение отдается просто сформулированным гипотезам, поскольку такие гипотезы легче изучить и, возможно, опровергнуть. Поэтому, например, изучить и оценить гипотезу о невозможности решения некоторой математической задачи проще, чем гипотезу о том, что система шифрования соответствует сложному определению безопасности. Еще одним преимуществом применения гипотез «более низкого уровня» (по сравнению с допущением о надежности конструкции) является то, что такие гипотезы могут быть использованы в других системах. И наконец, гипотезы низкого уровня обеспечивают *модульность*. Рассмотрим систему шифрования, чья надежность основывается на некотором принятом свойстве одного из ее структурных элементов. Если окажется, что базовый структурный элемент не соответствует указанному допущению, система шифрования все равно может быть реализована с использованием другого компонента, который считают соответствующим необходимым требованиям.

1.4.3 Принцип 3 – Доказательство безопасности

Два описанных выше принципа позволяют нам предоставить строгое *доказательство* того, что конструкция соответствует данному определению при конкретных, точно сформулированных допущениях. Такие доказательства особенно важны в криптографических условиях, в которых присутствует перехватчик, активно пытающийся «взломать» некоторую систему. Доказательства

безопасности предоставляют стопроцентную гарантию того, что в соответствии с определением и допущениями ни одному перехватчику не удастся взломать систему. Такой подход гораздо лучше не основанного на определенных принципах или эвристического метода решения задачи. Без доказательств того, что ни один противник, обладающий заданными ресурсами, не сможет взломать некоторую систему, мы лишь надеемся на нашу интуицию в том, что это именно так. Опыт показывает, что интуиция в криптографии и компьютерной безопасности разрушительна. Существует бесчисленное множество примеров недоказанных систем, которые были взломаны иногда сразу, иногда через несколько лет после их разработки.

Выводы: Строгий подход против ситуативного подхода по вопросу безопасности

Упор на определения, допущения и доказательства лежит в основе строгого подхода к криптографии, который отличается от неформального подхода классической криптографии. К сожалению, спонтанные, не основанные на определенных принципах решения разрабатываются и применяются теми, кто хочет быстро решить задачу или просто плохо осведомлен. Мы надеемся, что настоящая книга поможет лучше познакомиться со строгим подходом и осознать его важность при разработке доказуемо надежных систем.

Доказуемая безопасность и реальная безопасность

Сегодня значительная часть современной криптографии зиждется на обоснованных математических принципах. Но это не значит, что данная область также больше не является отчасти искусством. Строгий подход допускает возможность творческой изобретательности при создании определений, подходящих для современных приложений и сред, высказывании новых математических допущений или разработке новых примитивов, конструировании новейших систем и доказательстве их надежности. Конечно, всегда будет существовать и искусство *раскрытия* криптосистем, даже в случае их доказанной безопасности . Позже мы подробнее рассмотрим это утверждение.

Подход, используемый в современной криптографии, в корне изменил данную область. Он помогает обеспечить уверенность в стойкости криптографических систем в реальных условиях. Однако важно не переоценивать значение доказательства стойкости, которое всегда относится к рассматриваемому *определению* и выдвинутому(ым) *допущению(ям)*. Если гарантия безопасности не

---

[4]Здесь мы даже не рассматриваем возможность некорректной *реализации* системы. Неудачно разработанное шифрование в реальных условиях представляет собой серьезную проблему, однако, данная проблема находится за пределами охвата криптографии *как таковой*.

соответствует нуждам, или модель угроз не отражает реальных возможностей противника, тогда доказательство может оказаться бесполезным. Аналогичным образом, если допущение в основе доказательства оказывается ложным, тогда и все доказательство не имеет смысла.

Отсюда напрашивается вывод, что доказуемая стойкость системы не обязательно подразумевает ее реальную стойкость.[4] Некоторые считают это недостатком доказуемой стойкости, мы же относимся к этому оптимистично, поскольку таким образом демонстрируется эффективность подхода. Чтобы атаковать систему с доказуемой стойкостью в реальных условиях, достаточно сосредоточиться на определении (т.е. изучить, чем идеализированное определение отличается от реальных условий использования системы) или базовых допущениях (т.е. проверить их справедливость). В свою очередь, работа криптографов заключается в постоянном совершенствовании определений, чтобы последние максимально соответствовали реальным условиям, и изучении допущений с целью проверки их правильности. Доказуемая стойкость не прекращает издавна существующее противостояние между перехватчиком и защитником, однако, она служит основой, помогающей увеличить шансы защитника.

Ссылки и дополнительная литература

В настоящей главе мы изучили лишь немногие известные исторические шифры. Существует множество других шифров, представляющих исторический и математический интерес. За более подробной информацией мы рекомендуем читателю обратиться к учебникам Стинсона (Stinson) [168] или Трэппи (Trappe) и Вашингтона (Washington) [169]. Важная роль, которую криптография сыграла в истории, блестяще описана в книгах Кана (Kahn) [97] и Сингха (Singh) [163].

Принципы Керкгоффса освещены в [103, 104]. Шеннон (Shannon) [154], чью работу мы рассмотрим в следующей главе, был первым, кто начал применять в криптографии строгий подход, основанный на точных определениях и математических доказательствах.

Упражнения

Расшифруйте шифртекст, представленный в конце раздела по моноалфавитным подстановочным шифрам .

Дайте формальное определение алгоритмов Gen, Enc, и Dec моноалфавитного подстановочного шифра .

Дайте формальное определение алгоритмов Gen, Enc, и Dec шифра Виженера. (Примечание: существует несколько возможных вариантов для алгоритма Gen; выберите один.)

Выполните описанные в данной главе атаки на сдвиговый шифр и шифр Виженера.

Продемонстрируйте, что сдвиговый и подстановочный шифры, а также шифр Виженера традиционно раскрываются при использовании атаки на основе подобранного открытого текста. Открытый текст какого размера необходимо подобрать, чтобы восстановить ключ для каждого вида шифра?

Предположим, что перехватчику известно, что пароль пользователя abcd или bedg. Скажем, пользователь зашифровал свой пароль с помощью сдвигового шифра, и перехватчик видит получившийся шифртекст. Продемонстрируйте, каким образом перехватчик может определить пароль пользователя, или объясните, почему это невозможно.

Выполните предыдущее упражнение с шифром Виженера, последовательно используя периоды повторения 2, 3 и 4.

Сдвиговый и подстановочный шифры, а также шифр Виженера могут быть определены на алфавите ASCII, состоящем из 128 символов (а не на английском алфавите, состоящем из 26 символов).

(a) Дайте формальное определение для каждого вида систем, рассчитанных для такого случая.

(b) Обсудите, как показанные в настоящей главе атаки могут быть изменены для раскрытия каждой из измененных систем.

Глава 2

Абсолютная криптографическая стойкость

В предыдущей главе мы рассказали об исторических системах шифрования и показали, как они могут быть легко взломаны при небольших вычислительных затратах. В данной главе мы рассмотрим другую крайность и изучим системы с *доказуемой* надежности даже в случае неограниченных вычислительных возможностей перехватчика. Такие системы называются абсолютно *стойкий* . Кроме того, что мы дадим строгое определение понятия, мы изучим условия, при которых абсолютная стойкость может быть достигнута. (Начиная с этой главы, мы допускаем, что читатель знаком с основами теории вероятности. Соответствующие понятия рассмотрены в Приложении А.3.)

Материалы настоящей главы в некотором смысле больше относятся к «классической» криптографии, чем к «современной». Помимо того, что все представленные в данной главе материалы были разработаны до революции в области криптографии, прошедшей в середине 70-х и начале 80-х гг., конструкции, рассматриваемые нами в этой главе, опираются на первый и третий принципы, описанные в разделе 1.4. Иными словами, в основе этих систем лежат точные математические определения и строгие доказательства, хотя они не обязательно опираются на недоказанные вычислительные допущения. Совершенно очевидно, насколько выгодно избегать таких допущений, однако, мы увидим, что это, в свою очередь, накладывает определенные ограничения. Итак, помимо формирования подходящей основы для понимания базовых принципов современной криптографии настоящая глава поможет нам обосновать дальнейшее использование всех вышеупомянутых принципов.

В начале этой главы мы дадим определение безопасности и проанализируем системы с помощью вероятностных экспериментов, включающих в себя алгоритмы случайного выбора, на простом примере, когда стороны общаются, выбрав случайный ключ. Таким образом, прежде чем вернуться к вопросу криптографии как таковой, мы кратко обсудим вопрос генерации случайности, применяемой в криптографии.

Генерация случайности. На протяжении всей книги мы будем допускать, что стороны имеют доступ к неограниченному источнику по-настоящему произвольных битов. Откуда же берутся эти случайные биты на практике? В принципе, небольшое число случайных битов можно сгенерировать вручную,

например, просто подбрасывая правильную монету. Однако такой подход не очень удобен и не масштабен.

Современная г*енерация случайных чисел* осуществляется в два этапа. Сначала собирается массив данных с высокой энтропией. (Для наших целей формальное определение энтропии не требуется, достаточно считать энтропию степенью непредсказуемости.) Затем эти данные с высокой энтропией обрабатываются с целью получения последовательности почти по-настоящему произвольных битов. Этот второй этап необходим, поскольку данные с высокой энтропией не обязательно будут единообразны.

На первом этапе требуется некоторый источник непредсказуемых данных. Такие данные могут быть получены несколькими способами. Один из методов основывается на внешнем воздействии, например, простоях между сетевыми событиями, времени доступа к жесткому диску, нажатии клавиш или передвижении мыши пользователем, и т.д . Такие данные, скорее всего, будут далеки от единообразия, но если будет произведено достаточно измерений, то полученный массив данных будет обладать достаточной энтропией. Также используются более сложные подходы, которые, согласно замыслу, включают в себя генерацию случайных чисел, более тесно связанную с системой на уровне жесткого диска. В их основе лежат такие физические явления, как тепловые помехи, дробовый шум или радиоактивный распад. Недавно компания Intel (Интел) разработала процессор, включающий в себя цифровой генератор случайных чисел на микросхеме чипа и дающий специальные инструкции для доступа к полученным случайным битам (после их надлежащей обработки и получения по-настоящему произвольных битов, о которой расскажем далее).

Обработка, необходимая для «уравнивания» данных с высокой энтропией, с целью получения (почти) единообразных битов нетривиальна и кратко описывается в разделе 5.6.4. Здесь же мы только приведем простой пример, чтобы дать представление о том, что было сделано. Представьте, что наш массив с высокой энтропией был получен из последовательности подбрасываний *несимметричной* монеты, где вероятность выпадения «орла» равняется p, а вероятность выпадения «решки» равняется 1−p. (Однако мы допускаем, что результат любого подбрасывания монеты *не зависит* от всех остальных подбрасываний. На практике такое допущение обычно неверно.) Результат 1 000 таких подбрасываний монеты, конечно, будет обладать высокой энтропией, но будет не совсем единообразным. Мы можем получить равномерное распределение, рассматривая подбрасывания монеты по парам: если мы видим, что за «орлом» следует «решка», то мы выводим «0», а если за «решкой» следует «орел», то «1». (Если мы видим, что подряд выпали два «орла» или две «решки», то мы ничего не выводим, а просто переходим к следующей паре.) Вероятность того,

что результатом любой пары будет «0», составляет p• (1−p), что в точности соответствует вероятности того, что результатом любой пары будет «1». Таким образом, мы получаем равномерно распределенные выходные данные из нашего первоначального массива с высокой энтропией.

Необходимо крайне внимательно относиться к получению случайных битов, поскольку использование неэффективных генераторов случайных чисел часто может привести к тому, что хорошая криптосистема окажется уязвимой к атаке. Следует использовать генератор случайных чисел, *специально разработанный для криптографического использования*, а не генератор случайных чисел «общего назначения», не подходящий для криптографического использования. В частности, функция rand() в языке C в стандартной библиотеке stdlib.h не является криптографически надежной , а ее использование для криптографических параметров может иметь катастрофические последствия.

2.1 Определения

Начнем с того, что напомним и расширим синтаксис, введенный в предыдущей главе. Система шифрования определяется тремя алгоритмами Gen, Enc, и Dec, а также заданием *пространства* (конечного) *сообщения M* при |M|>1.[1] Алгоритм генерации ключа Gen - это вероятностный алгоритм, который выводит ключ k , выбираемый в соответствии с некоторой функцией распределения. Мы обозначаем (конечное) пространство ключей *K*, т.е. множество всех возможных ключей, которые могут быть выведены функцией Gen. Алгоритм шифрования Enc на входе принимает ключ $k \in K$ и сообщение $m \in M$ и на выходе выдает шифртекст c. Теперь допустим, что алгоритм шифрования является вероятностным (итак, Enck (m) может выводить различный шифртекст при неоднократном запуске), и запишем c ← Enck (m) для обозначения, возможно, вероятностного процесса, с помощью которого сообщение m будет зашифровано ключом k для пол учения шифртекста c. (Если алгоритм Enc является детерминированным, мы можем подчеркнуть это, написав c := Enck (m). Далее , мы также можем иногда использовать запись x ← S для обозначения единого отбора x из множества S.) Допустим, C обозначает множество всех возможных шифртекстов, которые могут быть выведены функцией Enck (m), для всех возможных выборов $k \in K$ и $m \in M$ (и для всех случайных выборов алгоритма Enc, если он случаен). Алгоритм расшифрования Dec принимает на входе ключ $k \in K$ и шифртекст $c \in C$, а на выходе выдает сообщение $m \in M$. Допустим *совершенную корректность*, т.е . для всех $k \in K$, $m \in M$ и любого шифртекста c, выведенного функцией Enck *(m)*, справедливо Deck (c) = *m* с вероятностью 1. Совершенная корректность подразумевает, что мы можем допустить, что алгоритм Dec является детерминированным, без потери общности, поскольку функция Deck *(c)* должна выводить один и тот же результат при каждом запуске. Таким образом, для обозначения процесса расшифрования шифртекста c с использованием ключа k для

получения сообщения m мы запишем m := Deck *(c)*.

В определениях и теоремах, представленных ниже мы ссылаемся на распределение вероятностей по *K, M,* и *C*. Распределение по K определяется запуском алгоритма Gen и получением вывода. (Так почти всегда и есть на самом деле, алгоритм Gen равномерно выбирает ключ из *K*, и мы фактически допускаем это без потери общности; см. Упражнение 2.1.) Предположим, что *K* - случайная переменная, указывающая значение ключа, выведенного алгоритмом Gen. Таким образом, для любого k ∈ K, Pr[*K* = *k*] обозначает вероятность того, что ключ, выведенный Gen, равняется k. Ана логичным образом допустим, что *M* - случайная переменная, обозначающая сообщение для шифрования, тогда Pr[M = m] означает вероятность того, что сообщение получит значение *m* ∈ *M*. Распределение вероятности сообщения не определено самой системой шифрования, но вместо этого отражает правдоподобную вероятность того, что стороны с помощью системы отправят разные сообщения, кроме того, перехватчик не будет уверен в том, что будет отправлено. Например , перехватчик может знать, что сообщение будет либо «attack today» (атакуй сегодня), либо «don't attack» (не атакуй). Перехватчик может даже узнать (другими способами), что вероятность сообщения, содержащего команду атаковать, составляет 0,7, а вероятность сообщения с командой не атаковать составляет 0,3. В таком случае мы имеем Pr[*M* = attack today] = 0,7 и Pr[*M* = don't attack] = 0,3.

K и *M* считаются независимыми, т.е. передаваемые сторонами сообщения не зависят от имеющегося у них ключа. Это имеет смысл еще и потому, что распределение по K определяется самой системой шифрования (она определена алгоритмом Gen), тогда как распределение M зависит от условий использования системы шифрования.

Фиксация системы шифрования и распределения по M определяет распределение по пространству шифртекста C, полученного при выборе ключа *k* ∈ K (в соо тветствии с алгоритмом Gen) и сообщения *m* ∈ M (в соответствии с заданным распределением), а затем вычисление шифртекста c ← Enck (m). Допустим, C - случайная переменная, определяющая полученный шифртекст. Таким образом, при *c* ∈ C для определения вероятности того, что шифртекст равняется заданному значению c, запишем Pr[C = c].

Пример 2.1

Рассмотрим простой пример для сдвигового шифра (см. Раздел 1.3). По определению мы имеем K = {0, ..., 25} при Pr[K = k] = 1/26 для каждого k ∈ K.

Скажем, было задано следующее распределение по M:

---

[1]Если |M| = 1, существует только одно сообщение и нет необходимости в коммуникации, не говоря уже о шифровании.

Pr[M = a] = 0,7 и Pr[M = z] = 0,3.

Какова вероятность того, что шифртекст равняется B? Для этого существует только две возможности: либо M = a и K = 1, либо M = z и K = 2. В результате независимости M и K мы имеем:

$$\Pr[M = \mathtt{a} \wedge K = 1] = \Pr[M = \mathtt{a}] \cdot \Pr[K = 1] = 0.7 \cdot \left(\frac{1}{26}\right).$$

Аналогичным образом, $\Pr[M = \mathtt{z} \wedge K = 2] = 0.3 \cdot \left(\frac{1}{26}\right)$. Следовательно,

$$\Pr[C = \mathtt{B}] = \Pr[M = \mathtt{a} \wedge K = 1] + \Pr[M = \mathtt{z} \wedge K = 2] = 0.7 \cdot \left(\frac{1}{26}\right) + 0.3 \cdot \left(\frac{1}{26}\right) = 1/26.$$

Также мы можем вычислить условные вероятности. Например, какова вероятность того, что сообщение a, учитывая, что мы имеем шифртекст B? Используя теорему Байеса (Теорема A.8), мы имеем

$$\Pr[M = \mathtt{a} \mid C = \mathtt{B}] = \frac{\Pr[C = \mathtt{B} \mid M = \mathtt{a}] \cdot \Pr[M = \mathtt{a}]}{\Pr[C = \mathtt{B}]} = \frac{0.7 \cdot \Pr[C = \mathtt{B} \mid M = \mathtt{a}]}{1/26}.$$

Обратите внимание, Pr[C = B | M = a] = 1/26, поскольку если M = a, тогда единственная возможность, при которой C = B, возникает, когда K = 1 (что происходит с вероятностью 1/26). Мы приходим к выводу, что Pr[M = a | C = B] = 0,7.

Пример 2.2

Еще раз рассмотрим сдвиговый шифр, но со следующим распределением по M:

Pr[M = kim] = 0.5, Pr[M = ann] = 0,2, Pr[M = boo] = 0,3.

Какова вероятность того, что C = DQQ? Единственная возможность, при которой это случится: если M = ann и K = 3 или M = boo и K = 2, что произойдет с вероятностью 0,2 • 1/26 + 0,3 • 1/26 = 1/52.

Итак, какова вероятность того, что было зашифровано ann, при условии, что наблюдаемый шифртекст DQQ? Вычисления по вышеупомянутой теореме Байеса дают: Pr[M = ann | C = DQQ] = 0,4.

Абсолютная стойкость. Теперь мы готовы дать определение понятию «*совершенная стойкость*». Представим противника, которому известно распределение вероятности по M, значит, перехватчику известна правдоподобная вероятность того, что будут отправляться разные сообщения. Данному противнику также известна используемая система шифрования. Единственное, что ему неизвестно, так это ключ, используемый сторонами. Один из честных участников

выбрал сообщение и зашифровал его, получив шифртекст, переданный другой стороне. Противник может *перехватить* сообщение, получив таким образом шифртекст. (Иными словами, это атака на основе шифртекста, когда перехватчик получает только один шифртекст.) Для того, чтобы система была совершенно секретной, изучение данного шифртекста не должно влиять на знания противника об отправленном в действительности сообщении. Иначе говоря, апостериорная вероятность того, что некоторое сообщение $m \in M$ было отправлено в зависимости от наблюдаемого шифртекста, не должна отличаться от априорной вероятности того, что m будет отправлено. Это означает, что шифртекст никак не отображает лежащий в его основе открытый текст и противник не получает никакой информации о зашифрованном открытом тексте. Формально:

ОПРЕДЕЛЕНИЕ 2.3 Система шифрования (Gen, Enc, Dec) с пространством сообщения M является совершенно секретной, если для каждого распределения вероятности по M, каждое сообщение $m \in M$ и каждый шифртекст $c \in C$, для которых $Pr[C = c] > 0$:

$Pr[M = m \mid C = c] = Pr[M = m]$.

(Условие $Pr[C = c] > 0$ является техническим и необходимым для предотвращения зависимости от события с нулевой вероятностью.)

Теперь мы дадим равноценную формулировку совершенной стойкости. Неформально, для настоящей формулировки необходимо, чтобы распределение вероятности шифртекста не зависело от открытого текста, т.е . для любых двух сообщений m, $m^r \in M$ распределение шифртекста при зашифрованном m должно быть тождественно распределению шифртекста при зашифрованном m^r. Формально, для каждого m, $m^r \in M$ и каждого $c \in C$

$$Pr[Enc_K(m) = c] = Pr[Enc_K(m^r) = c] \qquad (2.1)$$

(где вероятности основываются на выборе из K и любой случайности Enc). Из указанного следует, что шифртекст не содержит информации об открытом тексте и что невозможно отличить шифрование m от шифрования mr, поскольку в каждом случае распределения по шифртексту совпадают.

ЛЕММА 2.4 *Система шифрования (Gen, Enc, Dec) с пространством сообщения M является совершенно секретной тогда и только тогда, когда уравнение (2.1) верно для каждого m, $m^r \in M$ и каждого $c \in C$.*

ДОКАЗАТЕЛЬСТВО Покажем , что если заданное условие верно, то система совершенно секретна (обратную импликацию рассмотрим в Упражнении 2.4). Зафиксируем распределение по M, сообщению m и шифртексту c, для которого $Pr[C = c] > 0$. Если $Pr[M = m] = 0$, тогда мы обычно имеем:

$Pr[M = m \mid C = c] = 0 = Pr[M = m]$.

Итак, допустим, $\Pr[M = m] > 0$. Сначала отметим,

$$\Pr[C = c \mid M = m] = \Pr[\mathrm{Enc}K(M) = c \mid M = m] = \Pr[\mathrm{Enc}K(m) = c],$$

где первое равенство вытекает из определения случайной переменной C, а второе - из поставленного нами условия, что M равно m. Зададим $\delta_c \stackrel{\text{def}}{=} \Pr[\mathrm{Enc}_K(m) = c] = \Pr[C = c \mid M = m]$. Если условие леммы верно, тогда для каждого $m' \in M$ мы имеем $\Pr[\mathrm{Enc}K(m') = c] = \Pr[C = c \mid M = m'] = \delta_c$. Используя теорему Байеса (см. Приложение А.3), мы имеем:

$$\begin{aligned}
\Pr[M = m \mid C = c] &= \frac{\Pr[C = c \mid M = m] \cdot \Pr[M = m]}{\Pr[C = c]} \\
&= \frac{\Pr[C = c \mid M = m] \cdot \Pr[M = m]}{\sum_{m' \in \mathcal{M}} \Pr[C = c \mid M = m'] \cdot \Pr[M = m']} \\
&= \frac{\delta_c \cdot \Pr[M = m]}{\sum_{m' \in \mathcal{M}} \delta_c \cdot \Pr[M = m']} \\
&= \frac{\Pr[M = m]}{\sum_{m' \in \mathcal{M}} \Pr[M = m']} \quad = \quad \Pr[M = m],
\end{aligned}$$

где суммирование проводится по $m' \in M$ при $\Pr[M = m'] \neq 0$. Мы приходим к выводу, что для каждого $m \in M$ и $c \in C$, для которых $\Pr[C = c] > 0$, верно, что $\Pr[M = m \mid C = c] = \Pr[M = m]$, следовательно, система абсолютно секретна.

Совершенная неразличимость для противника. Данный раздел мы закончим, представив другое равнозначное определение совершенной стойкости. В основе этого определения лежит *эксперимент*, в ходе которого противник рассматривает шифртекст, не пытаясь его проанализировать или раскрыть, после чего пытается угадать, какое из двух возможных сообщений было зашифровано. Мы вводим данное понятие, поскольку оно служит отправной точкой для определения вычислительной стойкости в следующей главе. Действительно, на протяжении всей книги мы часто будет использовать такого рода эксперименты для определения безопасности.

В данном контексте мы рассмотрим следующий эксперимент: противник А сначала определяет два противоречащих друг другу сообщения $m0, m1 \in M$. Одно из этих двух сообщений было равномерно выбрано случайным образом и зашифровано с использованием случайного ключа. Получившийся шифртекст был передан А. В конце А выводит «догадку» о том, какое из двух сообщений было зашифровано. А *добивается успеха*, если догадка оказывается верна. Система шифрования является *совершенно неотличимой*, если ни один противник А не добьется успеха с вероятностью выше 1/2. (Обратите внимание, что для любой системы шифрования вероятность, что А достигнет успеха, составляет 1/2 при выводе равномерно распределенной догадки. Главное требование: ни один перехватчик не должен справиться лучше.) По дчеркиваем, вычислительные возможности А ничем не ограничиваются.

Формально, пусть П = (Gen, Enc, Dec) является системой шифрования с пространством сообщения M. Пусть A - это противник, который формально является лишь алгоритмом (отслеживающим состояние).

Обозначим содержание эксперимента $\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\Pi}$ следующим образом :

Эксперимент совершенной неразличимости для противника $\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\Pi}$:

1. П*ротивник A выводит пару сообщений m0, m1* ∈ M.

2. *С помощью алгоритма Gen генерируется ключ k и выбирается единообразный бит b* ∈ {0, 1}. *Вычисляется шифртекст c* ← *Enck (mb), который передается A. Мы рассматриваем c в качестве* анализируемого шифртекста.

3. *A выводит бит b*'.

4. *Результатом эксперимента является 1, если b'=b, иначе 0. Пишем* $P^{\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\Pi}}$ *= 1 если результатом является 1, тогда мы говорим, что A добился успеха.*

Как было отмечено ранее, обычно A достигает успеха с вероятностью 1/2, выводя случайную догадку. Для подтверждения совершенной неразличимости требуется, чтобы ни один A не сумел справиться лучше.

ОПРЕДЕЛЕНИЕ 2.5 Система шифрования П = (Gen, Enc, Dec) с пространством сообщения M является совершенно неразличимой, если для каждого A верно:

$$\Pr\left[\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\Pi} = 1\right] = \frac{1}{2}$$

Следующая лемма утверждает, что Определение 2.5 тождественно Определению 2.3. Доказать лемму требуется в Упражнении 2.5.

ЛЕММА 2.6 Система шифрования П является совершенной тогда и только тогда, когда является совершенно неразличимой.

Пример 2.7

Покажем, что шифр Виженера не является совершенно неразличимым, по крайней мере, по определенным параметрам. Пусть П обозначает шифр Виженера для пространства сообщения из двухсимвольных цепочек, где период повторения равномерно выбирается из {1, 2}. Чтобы показать, что П не является совершенно неразличимой, мы представим противника A, для которого

$\left[\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\Pi} = 1\right] > \frac{1}{2}$.

Противник A:

1. Выводит m0 = aa и m1 = ab.

2. После получения анализируемого шифртекста c = c1c2, выполняет следующее: если c1 = c2, выводит 0, иначе выводит 1. Вычисление

$$\Pr\left[\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\Pi} = 1\right] = \frac{1}{2}$$

банально, но просто.

$$\Pr\left[\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\Pi}=1\right]$$
$$=\frac{1}{2}\cdot\Pr\left[\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\Pi}=1\mid b=0\right]+\frac{1}{2}\cdot\Pr\left[\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\Pi}=1\mid b=1\right]$$
$$=\ 1/2\bullet\Pr[\text{A выводит }0\mid \text{b}=0]+1/2\bullet\Pr[\text{A выводит }1\mid \text{b}\ =1],\qquad(2.2)$$

где b - единообразный бит, определяющий, какое сообщение будет зашифровано. A выводит 0 тогда и только тогда, когда два символа шифртекста c = c1c2 равны. Когда b = 0 (значит зашифровано m0 = aa), тогда c1 = c2, если (1) выбран ключ периода 1 или (2) выбран ключ периода 2, и оба символа ключа равны. Первое происходит с вероятностью 1/2, а последнее происходит с вероятностью 1/2 • 1/26. Таким образом,

Pr[A выводит 0 | b = 0] = 1/2 + 1/2 • 1/26 ≈ 0,52.

Если b = 1, тогда c1 = c2 только если выбран ключ периода 2 и первый символ ключа на единицу больше второго символа ключа, что происходит с вероятностью 1/2 • 1/26. Таким образом,

Pr[A выводит 1 | b = 1] = 1 − Pr[A выводит 0 | b = 1] = 1 − 1/2 • 1/26 ≈ 0,98.

Подставляем в уравнение (2.2) и получаем:

$$\Pr\left[\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\Pi}=1\right]=\frac{1}{2}\cdot\left(\frac{1}{2}+\frac{1}{2}\cdot\frac{1}{26}+1-\frac{1}{2}\cdot\frac{1}{26}\right)=0.75>\frac{1}{2},$$

и система не является совершенно неразличимой. ♦

2.2 Шифр Вернама

В 1917 году Гилберт Вернам запатентовал *абсолютную криптографическую стойкость*, названную шифром Вернама (или одноразовым блокнотом). Когда Вернам предложил свою систему, доказательств ее абсолютной стойкости не существовало, впрочем, как и самого понятия абсолютной стойкости. Тем не менее, примерно 25 лет спустя Клод Шеннон ввел определение абсолютной стойкости и продемонстрировал, что шифр Вернама достигает такого уровня безопасности.

КОНСТРУКЦИЯ 2.8

Зафиксируем целое число $l > 0$. Пространство сообщения M, шифртекста C все равны

- Gen: алгоритм генерации ключей выбирает ключ из K в соответствии с равномерным распределением пространства выбрана как и ключ с вероятностью, составляющей точно 2^{-l}).
- Enc: заданы ключ $k \in \{0,1\}^l$ и сообщение $m \in \{0,1\}^l$, алгоритм шифрования выводит шифртекст $c := k \oplus m$.
- Dec: заданы ключ $k \in \{0,1\}^l$ и шифртекст $c \in \{0,1\}^l$,

Схема шифрования Вернама.

Для описания системы пусть $a \oplus b$ означает *побитовое исключающее* ИЛИ (XOR) двух двоичных последовательностей a и b (т.е ., если $a = a1 \bullet\bullet\bullet aA$ и $b = b1 \bullet\bullet\bullet bA$ являются строками A-битов, тогда $a \oplus b$ является строкой А-битов, заданных $a1 \oplus b1 \bullet\bullet\bullet aA \oplus bA$). Кл юч в системе шифрования Вернама - это равномерная строка той же длины, что и сообщение , а шифртекст вычисляется путем простого объединения ключа и сообщения операцией исключающего ИЛИ. Формальное определение дано в Конструкции 2.8. Перед обсуждением стойкости системы мы проверим ее корректность: для каждого ключа k и каждого сообщения m верно, что Deck (Enck (m)) = $k \oplus k \oplus m = m$, таким образом, шифр Вернама представляет собой надежную систему шифрования.

Опираясь на лемму 2.4 и тот факт, что шифртекст равномерно распределен вне зависимости от зашифрованного сообщения, можно с легкостью доказать абсолютную стойкость шифра Вернама. Приведем доказательство, основанное непосредственно на первоначальном определении.

ТЕОРЕМА 2.9 *Система шифрования Вернама обладает абсолютной криптографической стойкостью .*

ДОКАЗАТЕЛЬСТВО Сначала вычислим Pr[$C = c \mid M = m^r$] для произвольных $c \in C$ и $mr \in M$. Для шифра Вернама

$$\Pr[C=c|M = m^r] = \Pr[\mathrm{Enc}K(m^r)=c] = \mathrm{r}[m^r \oplus K = c] = \Pr[K = m^r \oplus c] = 2^{-\mathrm{A}},$$

где конечное равенство верно, поскольку ключ K является равномерно распределенной строкой A-битов. Зафиксируем любое распределение по M. Для всех $c \in C$, мы имеем:

$$\begin{aligned}\Pr[C = c] &= \sum_{m' \in \mathcal{M}} \Pr[C = c \mid M = m'] \cdot \Pr[M = m'] \\ &= 2^{-\ell} \cdot \sum_{m' \in \mathcal{M}} \Pr[M = m'] \\ &= 2^{-\ell},\end{aligned}$$

где сумма превышает $m^r \in M$ при Pr[$M = m^r$] $f= 0$. Из теоремы Байеса имеем:

$$\begin{aligned}\Pr[M = m \mid C = c] &= \frac{\Pr[C = c \mid M = m] \cdot \Pr[M = m]}{\Pr[C = c]} \\ &= \frac{2^{-\ell} \cdot \Pr[M = m]}{2^{-\ell}} \\ &= \Pr[M = m].\end{aligned}$$

Мы приходим к выводу, что шифр Вернама обладает абсолютной криптографической стойкостью.

В середине XX века разведывательное управления некоторых стран использовали шифр Вернама для шифрования особо важных сообщений. Вероятно,

наиболее известным применением шифра Вернама является защищенная с его помощью «горячая линия» между Вашингтоном и Москвой времен «холодной войны», когда правительства США и СССР обменивались невероятно длинными ключами, перевозимыми надежными курьерами в портфелях, полных документов с записанными в них случайными символами.

Несмотря на вышесказанное, шифр Вернама используется редко из-за количества его недостатков. Самым существенным его недостатком является то, что *длина ключа должна совпадать с длиной сообщения*.[2] Таким образом ограничивается эффективность системы при пересылке очень длинных сообщений (сложно безопасно передать и хранить очень длинный ключ) и создает определенные трудности, когда стороны не могут заранее предсказать (верхнюю границу), какой длины будет сообщение.

Кроме того, шифр Вернама, как следует из его другого названия (одноразовый блокнот), *безопасен, только когда используется один раз* (с одним ключом). И хотя мы еще не дали определение криптографической стойкости при шифровании нескольких сообщений, легко заметить, что шифрование более чем одного сообщения одним ключом приводит к большой утечке информации. В частности, скажем , два сообщения m, mr зашифрованы с использованием одного ключа k. Противник, получивший $c = m \oplus k$ и $c^r = m^r \oplus k$, может вычислить

$$c \oplus c^r = (m \oplus k) \oplus (m^r \oplus k) = m \oplus mr$$

и таким образом узнает «исключающее ИЛИ» двух сообщений или, что равнозначно, различия между двумя сообщениями. Хотя это может показаться не очень существенным, но этого достаточно, чтобы исключить любую возможность считать систему шифрования двух сообщений одним и тем же ключом совершенно криптографически надежной. Более того, если сообщения соответствуют тексту на естественном языке, то получив «исключающее ИЛИ» двух достаточно длинных сообщений, можно провести частотный анализ (сложнее, чем описанный в предыдущей главе) и восстановить сами сообщения. Интересным историческим примером для нас служит *проект «Венона»* (VENONA), в ходе которого разведка США и Соединенного Королевства сумела расшифровать шифровки Советского Союза, по ошибке зашифровываемые повторяющимися ключами шифра Вернама на протяжении нескольких десятилетий.

2.3 Ограничения абсолютной стойкости

Предыдущий раздел мы завершили тем, что отметили некоторые недостатки системы шифрования Вернама. Здесь же мы покажем, что эти недостатки не являются особенностями той системы шифрования, а, напротив, являются *неотъемлемыми* ограничениями абсолютной стойкости. Более того, мы докажем, что длина пространства ключа любой абсолютно стойкой системы шифрования должна по крайней мере совпадать с длиной пространства сообщения. Если все ключи имеют одну

длину, а пространство сообщения состоит из всех строк некоторой фиксированной длины, это подразумевает, что ключ имеет по крайней мере такую же длину, что и сообщение. В частности, длина ключа шифра Вернама является оптимальной. (Другое ограничение, а именно однократное использование одного ключа, также является неотъемлемым требованием совершенной секретности. См. Упражнение 2.13.)

ТЕОРЕМА 2.10 *Если* (Gen, Enc, Dec) *является схемой абсолютной криптографической стойкостью с пространством сообщения M и пространством ключа K, тогда* $|K| \geq |M|$.

ДОКАЗАТЕЛЬСТВО Покажем, что если $|K| < |M|$, тогда система не может быть совершенно криптографически надежной. Допустим , $|K| < |M|$. Рассмотрим равномерное распределение по M, пусть $c \oplus C$ - это шифртекст, встречающийся с ненулевой вероятностью. Пусть M(c) - это множество всех возможных сообщений, являющихся возможной расшифровкой c, из чего следует: что

$$M(c) \stackrel{\text{def}}{=} \{m \mid m = \mathrm{Dec}_k(c) \text{ для некоторого } k \in K\}.$$

Очевидно, $|M(c)| \leq |K|$. (Напомним , что мы можем допустить, что алгоритм Dec является детерминированным.) Если $|K| < |M|$, существует некоторое $m^r \in M$ такое, что $m^r \not\in M(c)$. Но тогда

$$\Pr[M = m^r \mid C = c] \neq 0 = \Pr[M = m^r],$$

из чего следует, что система не является абсолютно криптографически стойкой.

Существование абсолютной криптографической стойкости с более короткими ключами? Вышеупомянутая теорема иллюстрирует неотъемлемое ограничение систем, достигших совершенной стойкости. Тем не менее, время от времени некоторые люди заявляют, что они изобрели совершенно новую «невзламываемую» систему шифрования, обладающую стойкостью шифра Вернама, но не использующую для шифрования ключ той же длины, что и сообщение. Приведенное выше доказательство свидетельствует о том, что такие заявления не могут быть правдивыми. Любой, кто делает подобные заявления либо совершенно не разбирается в криптографии, либо в открытую врет.

2.4 * Теорема Шеннона

В своей работе по абсолютной криптографической стойкости Шеннон также привел описание совершенно надежных систем шифрования. В этом описании говорилось, что при определенных условиях алгоритм генерации ключа Gen должен *равномерно* выбрать ключ из множества возможных ключей (как при

[2]Это не значит, что шифр Вернама бесполезен, поскольку двум сторонам может быть проще обменяться ключами в какой-то определенный момент до пересылки сообщения.

шифровании Вернама); более того, для каждого сообщения m шифртекста c существует *уникальный ключ*, преобразующий *m* в *c* (опять-таки как при шифровании Вернама). Помимо того, что она интересна сама по себе, данная теорема является полезным инструментом, доказывающим (или опровергающим) совершенную криптографическую стойкость предложенных систем. Обсудим это чуть позже, после ее доказательства.

Как уже было сказано, теорема допускает, что |M| = |K| = |C|, иными словами, размер всех множеств открытых текстов, ключей и шифртекстов совпадает. Мы уже знаем, что обязательным условием совершенной криптографической стойкости является |K| ≥ |M|. Легко заметить, что для правильного расшифрования необходимо: |C| ≥ |M|. Получается, что в некотором смысле системы шифрования при |M| = |K| = |C| являются «оптимальными».

ТЕОРЕМА 2.11 *(Теорема Шеннона) Пусть (Gen, Enc, Dec) - это система шифрования с пространством сообщения M, для которого* |M| = |K| = |C|. *Система шифрования является совершенно криптографически стойкой тогда и только тогда, когда:*

1.Каждый ключ k∈K выбран алгоритмом Gen с (равной) вероятностью 1/|K|.

2. Для каждого m ∈ M и каждого c ∈ C существует такой уникальный ключ k ∈ K, что Enck (m) выводит c.

ДОКАЗАТЕЛЬСТВО На уровне интуиции доказательство выглядит следующим образом: Для понимания того, что заданные условия подразумевают абсолютную криптографическую стойкость, обратите внимание на то, что условие 2 означает, что любой шифртекст c может быть результатом шифрования любого возможного открытого текста m, поскольку существует некоторый ключ *k*, преобразующий *m* в *c*. Поскольку существует такой *уникальный* ключ, а каждый ключ выбирается с равной вероятностью, то совершенная криптографическая стойкость доказывается как и для шифра Вернама. И наоборот, совершенная криптографическая стойкость автоматически подразумевает, что для каждого m и c существует по крайней мере один ключ, преобразующий m в c. Более того, из |M| = |K| = |C| вытекает вывод, что для каждого m и c существует только один такой ключ. Исходя из этого, каждый ключ должен быть выбран с равной вероятностью, иначе абсолютная криптографическая стойкость не будет достигнута. Формально же доказательство выглядит так:

Для удобства допустим, что алгоритм Enc детерминирован. (Здесь можно говорить об этом без ущерба для общности.) Сначала докажем, что если система шифрования удовлетворяет условиям 1 и 2, то она является абсолютно криптографически надежной. В сущности, доказательство такое же, как и доказательство абсолютной стойкости шифра Вернама, поэтому мы будем довольно кратки. Зафиксируем произвольные *c* ∈ *C* и *m* ∈ M. Пусть *k* - уникальный ключ,

заданный условием 2, для которого верно Enck (m) = c. Тогда ,

Pr[C = c | M = m] = Pr[K = k] = 1/|K| ,

где конечное равенство верно по условию 1. Таким образом ,

$$\Pr[C = c] = \sum_{m \in \mathcal{M}} \Pr[\mathsf{Enc}_K(m) = c] \cdot \Pr[M = m] = 1/|\mathcal{K}|.$$

Это верно для любого распределения по M. Таким образом, для любого распределения по M, любого $m \in M$ при Pr[$M = m$] f= 0 и любого $c \in C$ мы имеем:

$$\begin{aligned}\Pr[M = m \mid C = c] &= \frac{\Pr[C = c \mid M = m] \cdot \Pr[M = m]}{\Pr[C = c]} \\ &= \frac{\Pr[\mathsf{Enc}_K(m) = c] \cdot \Pr[M = m]}{\Pr[C = c]} \\ &= \frac{|\mathcal{K}|^{-1} \cdot \Pr[M = m]}{|\mathcal{K}|^{-1}} \;=\; \Pr[M = m],\end{aligned}$$

и система является абсолютно криптографически стойкой.

А теперь докажем обратную теорему и допустим, что система шифрования является абсолютно стойкой . Покажем, что условия 1 и 2 верны. Зафиксируем произвольное $c \in C$. Должно существовать некоторое сообщение m*, для которого [EncK (m*) = c] f= 0. Лемма 2.4 тогда подразумевает, что Pr[EncK (m) = c] f= 0 для каждого $m \in M$. Иными словами, если мы допустим, что M = {m1, m2, . . .}, тогда для каждого $mi \in M$ мы имеем непустое множество ключей Ki ∈ K такое, что Enck (mi) = c тогда и только тогда, когда k ∈ Ki. Более того, когда i f= j, тогда Ki и Kj не должны пересекаться, иначе утверждение не верно. Поскольку |K| = |M| , мы видим, что каждое Ki содержит только один ключ ki, как того и требует условие 2. Теперь лемма 2.4 показывает, что для любого mi, mj ∈ M мы имеем

Pr[K = ki] = Pr[EncK (mi) = c] = Pr[EncK (mj) = c] = Pr[K = kj].

Поскольку это верно для всех 1 ≤ i, j≤|M| = |K|, и ki f= kj для i f= j, следовательно, каждый ключ, выбран с вероятностью 1/|K|, как того и требует условие 1.

Теорема Шеннона полезна для принятия решения о абсолютной криптографической стойкости заданной системы. Условие 1 легко проверяется, а условие 2 может быть подтверждено (или опровергнуто) без необходимости вычислять все вероятности (по сравнению с непосредственной работой по определению 2.3). Например, совершенная стойкость шифра Вернама обычно доказывается с помощью теоремы Шеннона. Однако обращаем ваше внимание на то, что теорема применяется, когда |M| = |K| = |C|.

Ссылки и дополнительная литература

Изобретение шифра по типу одноразового блокнота обычно приписывается

запатентовавшему его Вернаму [172], однако, недавние исторические исследования [25] показали, что шифр был изобретен примерно на 35 лет раньше. Анализ шифра Вернама пришлось отложить до появления новаторской работы Шеннона [154], который ввел понятие абсолютной криптографической стойкости. В настоящей главе мы рассмотрели абсолютную криптографическую стойкость . Некоторые другие криптографические задачи могут также решаться с помощью «абсолютной» безопасности . Важнейшим примером может служить задача аутентификации сообщения, целью которой является предотвратить (неопределяемое) изменение противником сообщения, направленного одной стороной другой. Более подробно мы рассмотрим эту проблему в разделе 4.6 главы 4 в ходе обсуждения «абсолютно криптографически стойких» аутентификаций сообщения.

Упражнения

Докажите, что при переопределении пространства ключа мы можем допустить, что алгоритм генерации ключей Gen выбирает ключ равномерно произвольно из пространства ключей без изменения Pr[C = c | M = m] для любых m, c.

Подсказка: Определите, что пространство ключей - это множество всех возможных случайных лент для случайного алгоритма Gen.

Докажите, что при переопределении пространства ключа мы можем допустить, что алгоритм Enc является детерминированным без изменения Pr[C = c | M = m] для любых m, c.

Докажите или опровергните: Система шифрования с пространством сообщения M является совершенно криптографически надежной тогда и только тогда, когда для каждого распределения вероятности по M и каждого c0, c1 $\in$ C мы имеем Pr[C = c0] = Pr[C = c1].

Докажите утверждение, обратное лемме 2.4.

Докажите лемму 2.6.

Определите, является ли каждая из приведенных систем шифрования совершенно криптографически надежной. Обоснуйте свой ответ в каждом из случаев.

(a) Пространство сообщения M={0, ..., 4}. Алгоритм Gen выбирает унифицированный ключ из пространства ключей {0, . . . , 5}. Enck (m) выдает [k + m mod 5], a Deck (c) выдает [c − k mod 5].

(b) Пространство сообщения M = {m $\in$ {0, 1}A | последний бит m равен 0}. Алгоритм Gen унифицированный ключ из {0, 1}A−1. Enck (m) выдает шифртекст m $\oplus$ (k''0), a Deck (c) выводит c $\oplus$ (k''0).

При использовании шифра Вернама с ключом k = 0A , мы имеем Enck (m) = k $\oplus$ m = m и отправляется незашифрованное сообщение! В следствие этого было предложено изменить шифр Вернама, используя для шифрования k f= 0^A (т. е.

заставить алгоритм Gen равномерно выбирать k из множества ненулевых ключей длиной A). Повлияло ли это на совершенную надежность системы? Объясните.

Пусть П обозначает шифр Виженера, где пространство сообщения состоит из всех трехсимвольных строк (английский алфавит), а ключ сгенерирован путем равномерного выбора периода t из {1, 2, 3}, после чего допускалось, что ключ - это равномерная строка длиной t.

(a) Определите A следующим образом: A выводит m0 = aab и m1 = abb. После задания шифртекста c A выводит 0, если первый символ c совпадает со вторым символом c, иначе выводит 1. Вычислите $\Pr\left[\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\Pi} = 1\right]$

(b) Создайте и проанализируйте противника A^r, для которого $\Pr\left[\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\Pi} = 1\right]$ больше вашего ответа из части (a).

В данном упражнении мы рассматриваем разные условия, при которых шифры сдвига, моноалфавитной подстановки и шифр Виженера являются совершенно криптографически надежными:

(a) Докажите, что если зашифровать один символ, тогда сдвиговый шифр является совершенно криптографически надежным.

(b) Каково же наибольшее значение пространства сообщения M, для которого моноалфавитный шифр подстановки обеспечит совершенную стойкость?

(c) Докажите, что шифр Виженера, использующий (фиксированный) период t, является абсолютно стойким при шифровании сообщения длиной t.

Согласуйте это с атаками, представленными в предыдущей главе.

Докажите, что система, удовлетворяющая определению 2.5, должна иметь $|K| \geq |M|$ без использования леммы 2.4. В частности, пусть П - это произвольная система шифрования с $|K| < |M|$. Представьте A, для которого $\Pr[\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\Pi} = 1] > \frac{1}{2}$

Подсказка: Возможно, легче допустить, что A случайна .

Допустим, нам только требуется, чтобы система шифрования (Gen, Enc, Dec) с пространством сообщения M удовлетворяла следующему условию: Для всех $m \in M$, мы имеем $\Pr[\mathrm{DecK}(\mathrm{EncK}(m)) = m] \geq 2^{-t}$. (Данная вероятность переносится на выбор ключа таким же образом, как и любая произвольность, использованная во время шифрования.) Покажите , что совершенная стойкость может быть достигнута при $|K| < |M|$, когда $t \geq 1$. Докажите существование нижней границы размера K исходя из t.

Пусть $\varepsilon \geq 0$ будет постоянным. Предположим, что система шифрования является ε-абсолютно стойкой , если для каждого противника A верно:

$$\Pr\left[\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\Pi} = 1\right] \leq \frac{1}{2} + \varepsilon$$

(Сравните с определением 2.5) Покажите, что ε-абсолютная стойкость может

быть достигнута при $|K| < |M|$, когда $\varepsilon > 0$. Докажите существование нижней границы размера K исходя из ε

В данной задаче мы рассматриваем определения совершенной стойкости системы для шифрования двух сообщений с использованием одного ключа. Здесь мы рассматриваем распределение по парам сообщений из пространства сообщения M. Пусть M1, M2 - произвольные переменные, определяющие первое и второе сообщение соответственно. (Подчеркиваем, что мы не допускаем того, что эти произвольные переменные независимы.) Генерируем (один) ключ k, отбираем пару сообщений (m1, m2) в соответствии с заданным распределением, после чего вычисляем шифртексты $c1 \leftarrow Enc_k(m1)$ и $c2 \leftarrow Enc_k(m2)$, что обуславливает распределение по парам шифртекстов, пусть C1, C2 – соответствующие произвольные переменные.

(a) Скажем, система шифрования (Gen, Enc, Dec) является абсолютно стойкой для двух сообщений, если для всех распределений по $M \times M$ все $m1, m2 \in M$ и все шифртексты $c1, c2 \in C$ при $Pr[C1 = c1 \wedge C2 = c2] > 0$:

$$Pr[M1 = m1 \wedge M2 = m2 \mid C1 = c1 \wedge C2 = c2] = Pr[M1 = m1 \wedge M2 = m2].$$

Докажите, что ни одна система шифрования не соответствует данному определению.

Подсказка: Возьмите $c1 = c2$.

(b) Скажем, система шифрования (Gen, Enc, Dec) является совершенно надежной для двух отдельных сообщений, если для всех распределений по $M \times M$, где первое и второе сообщения обязательно являются разными (т.е ., распределения по парам разных сообщений), все $m1, m2 \in M$, и все $c1, c2 \in C$ при $Pr[C1 = c1 \wedge C2 = c2] > 0$:

$$Pr[M1 = m1 \wedge M2 = m2 \mid C1 = c1 \wedge C2 = c2] = Pr[M1 = m1 \wedge M2 = m2].$$

Покажите систему шифрования, которая доказуемо соответствует данному определению.

Подсказка: Предложенная вами система шифрования не обязательно должна быть эффективной, хотя эффективное решение возможно .

Часть II

(Симметричная) Криптография с закрытым ключом

Глава 3

Шифрование с закрытым ключом

В предыдущей главе мы рассмотрели некоторые фундаментальные ограничения абсолютной криптографической стойкости. В данной главе мы начнем изучение современной криптографии с введения более слабого (но обоснованного) понятия *вычислительной* стойкости. Затем мы покажем, как это определение можно использовать, чтобы обойти ранее показанные результаты невероятности и, в частности, как короткий ключ (скажем, длиной 128 бит) может быть использован для шифрования множества длинных сообщений (скажем, общим размеров в несколько гигабайт).

Также мы изучим фундаментальное понятие *«псевдослучайность»*, которое означает, что нечто может «выглядеть» случайным, но на самом деле таким не является. Это понятие лежит в основе современной криптографии, а также используется и подразумевается в областях за ее пределами.

3.1 Расчетная стойкость

В Главе 2 мы ввели понятие «абсолютная стойкость». И хотя абсолютная стойкость - цель многообещающая, она еще и необоснованно высокая. Абсолютная стойкость предусматривает *абсолютное отсутствие утечки информации о зашифрованном сообщении*, даже если подслушивающая сторона *обладает неограниченными вычислительными возможностями*. Для практических же целей система шифрования будет считаться надежной, если она допускает *ничтожно малую* утечку информации в случае вмешательства подслушивающей стороны с *ограниченными вычислительными возможностями*. Например, система, допускающая утечку информации с максимальной вероятностью равной 2−60 при вмешательстве подслушивающей стороны, обладающей вычислительной мощностью равной 200 годам вычислений на самом быстром из доступных суперкомпьютеров, вполне подходит для использования в реальных условиях. Стойкость, в соответствии с определением учитывающая предел вычислительных возможностей перехватчика и допускающая малую вероятность провала, называется в*ычислительной* и отличается от *информационно-теоретических* по своей природе понятий (например, абсолютная стойкость). Сейчас вычислительная стойкость является реальным направлением, в котором стойкость определяется для всех криптографических целей.

Подчеркиваем, что хотя мы и отказываемся от достижения абсолютной стойкости, это не означает, что мы отходим от строгого математического подхода.

Определения и доказательства все еще играют важную роль. Единственное отличие - мы рассматриваем более слабые (но все еще эффективные) определения стойкости.

Расчетная стойкость объединяет в себе два ослабления по сравнению с информационно-теоретическими понятиями стойкости (в случае шифрования, оба эти ослабления необходимы для выхода за пределы ограничений абсолютной стойкости, описанных в предыдущей главе):

1. *Стойкость обеспечивается только против эффективных* противников, *работающим в течение правдоподобного количества времени*. Это означает, что если перехватчик обладает достаточным количеством времени (или достаточными вычислительными возможностями), он может взломать систему. Если мы сумеем создать систему, для взламывания которой потребуются ресурсы, превышающие ресурсы реального злоумышленника, для всех практических целей мы можем считать систему невзламываемой.

2. *Противники потенциально могут добиться успеха (т.е. стойкость потенциально может быть нарушена)* с очень малой вероятностью. Если мы сможем свести такую вероятность к достаточному минимуму, нам не придется переживать из-за нее.

Для построения значимой теории нам необходимо точно определить вышеуказанные ослабления. Для этой цели существует два общих подхода: *конкретный подход* и *асимптотический подход* - оба из которых описываются далее.

3.1.1 Конкретный подход

Конкретный подход к вычислительной стойкости измеряет стойкость криптографической системы, явно ограничивая максимальную вероятность успеха любого (случайного) противника, запущенного в течение некоторого указанного промежутка времени или, точнее, затрачивающего некоторый указанный объем вычислительных возможностей. Таким образом, конкретное определение стойкости принимает приблизительно следующую форма:

> *Система является (t, ε)-стойкой, если противник, запущенный в течение максимального времени t успешно взламывает систему с максимальной вероятностью ε.*

(Конечно, вышеприведенное утверждения служит общим шаблоном и для того, чтобы оно имело смысл, нам необходимо точно определить, что значит «взломать» рассматриваемую систему.) Например, кто-то может обладать системой, гарантирующей, что противник, работающий в течение максимум 200 лет и использующий самый быстрый из доступных суперкомпьютер, сумеет взломать систему с вероятностью лучше, чем 2−60. Или может быть удобнее измерять время работы в циклах центрального процессора и создать систему, которую ни один противник, использующий максимум 2^{80} циклов централь-

ного процессора, сумеет взломать систему с вероятностью лучше, чем 2^{-60}.

Полезно прочувствовать большие значения t и малые значения ε, типичные для современных криптографических систем.

Пример 3.1

Обычно допускается, что современные системы шифрования с закрытым ключом обеспечивают почти оптимальную надежность следующим образом: когда длина ключа составляет n (значит, пространство ключа составляет 2^n), противнику, запущенному в течение времени t (измеряется, скажем, в машинных циклах), удается взломать систему с максимальной вероятностью $ct/2^n$, для некоторого постоянного значения c. (Это соответствует простому перебору пространства ключа и допускает, что никакой анализ не проделывался.)

Допустим для простоты, что c = 1, ключ длиной n = 60 обеспечивает необходимую стойкость против противника с обычным настольным компьютером. Действительно, на процессоре с частотой 4 ГГц (выполняет 4×10^9 циклов в секунду) 2^{60} циклов центрального процессора займут $2^{60}/(4\times10^9)$ секунд, или около 9 лет. Однако самый быстрый суперкомпьютер на момент написания нами книги может выполнить 2×10^{16} операций с плавающей точкой за секунду, и выполнение 2^{60} таких операций займет у него около одной минуты . Взять n = 80 было бы более благоразумным решением, поскольку даже у вышеупомянутого компьютера на выполнение 2^{80} операций уйдет два года.

(Вышеупомянутые числа приведены только для наглядности: на практике c > 1, а другие некоторые факторы (время, необходимое для доступа к памяти и возможность параллельного вычисления с помощью сетевых компьютеров) могут значительно повлиять на атаки методом перебора.)

Тем не менее, сегодня рекомендуемая длина ключа может быть n = 128. Разница между 2^{80} и 2^{128} является мультипликативным коэффициентом, равным 2^{48}. Чтобы иметь представление о том, насколько это большое число, необходимо отметить, что по оценке физиков, количество секунд, прошедших после Большого Взрыва составляет порядка 2^{58}.

Если вероятность того, что перехватчик успешно восстановит зашифрованное сообщение за один год, составляет самое большее 2^{-60}, тогда такова же вероятность, что в этот же временной промежуток в отправителя и получателя ударит молния.

Вероятность того, что в любую секунду произойдет событие, происходящее один раз в сто лет, составляет примерно 2^{-30}. Нечто, происходящее каждую секунду с вероятностью 2^{-60}, случится с вероятностью в 230 раз меньше, т.е . случается приблизительно один раз в 100 млрд лет.♦

Конкретный подход важен на практике, поскольку конкретные гарантии - это

именно то, что интересует пользователей криптографических систем. Тем не менее, сложно предоставить точные и конкретные гарантии. Более того, необходимо очень осторожно интерпретировать заявления о конкретной стойкости. Например, заявление о том, что ни один работающий в течение 5 лет противник не сможет взломать данную систему с вероятностью больше ε, вызывает следующие вопросы: Какой тип вычислительной мощности (например, настольный ПК, суперкомпьютер, сеть из сотен компьютеров) подразумевается? Учитывает ли он возможный прогресс в области вычислительной мощности, которые по закону Мура удваиваются каждые 18 месяцев? Допускает ли оценка использование «типовых» алгоритмов или внедрение ПО, оптимизированного для атаки? Кроме того, такие гарантии ничего не говорят о вероятности успеха противника, работающего в течение 2 лет (за исключением того факта, что это может произойти с максимальной вероятностью ε), и ничего не говорится о вероятности успеха противника, работающего в течение 10 лет.

3.1.2 Асимптотический подход

Как уже отчасти было сказано выше, существуют некоторые технические и теоретические сложности, связанные с использованием конкретного подхода в вопросах стойкости системы. Эти вопросы должны решаться на практике, но если конкретная стойкость не является безотлагательной проблемой, удобнее использовать *асимптотический подход* к стойкости. Именно такой подход и применяется в данной книге. Этот подход, уходящий своими корнями в теорию сложности, вводит целочисленный *параметр безопасности n*, который параметризует криптографические системы, а также задействованные стороны: честных участников и перехватчика. Когда честные участники запускают систему (т.е. генерируют ключи), они выбирают некоторое значение n в качестве параметра безопасности. В целях настоящего обсуждения условимся, что параметр безопасности соответствует длине ключа. Допустим, что параметр безопасности известен противнику, атакующему систему, теперь мы рассматриваем время работы противника, а также вероятность успеха, поскольку функции параметра вероятности больше функций конкретных чисел. Тогда :

1. Мы приравниваем «эффективных противников» к случайным (т.е. вероятностным) алгоритмам, работающим в течение *полиномиального* времени от n. Это значит, что существует некоторый многочлен p, такой что противник работает в течение максимум p(n), когда параметр безопасности равен n. Для реальной эффективности нам также необходимо, чтобы честные участники тоже действовали в полиномиальном времени, однако, подчеркнем, что противник может обладать большими возможностями (и действовать значительно дольше), чем честные участники.

2. Мы приравниваем понятие «малая вероятность успеха» к вероятности успеха *меньше, чем любой обратный многочлен от n* (см. Определение 3.4).

Такие вероятности называются пренебрежимо малыми.

Пусть ppt означает «вероятностное полиномиальное время». Тогда определение асимптотической стойкости примет следующую общую форму:

> *Система является надежной, если любой ppt противник может взломать систему с пренебрежимо малой вероятностью.*

Такое понятие безопасности является асимптотичным, поскольку безопасность зависит от поведения системы при достаточно больших значениях n. Поясним это с помощью следующего примера.

Пример 3.2

Скажем, у нас есть *асимптотически* надежная система. Тогда возможно, что противник, работающий в течение n^3 минут, сумеет «взломать систему» с вероятностью $2^{40} \cdot 2^{-n}$, что является пренебрежимо малой функцией по n. При $n \leq 40$ это означает, что противник работает в течение 40^3 минут (около 6 недель) может взломать систему с вероятностью 1, поэтому такие значения n неприменимы. Даже при n = 50 противник, работающий в течение 50^3минут (около 3-х месяцев) может взломать систему с вероятностью около 1/1000, что тоже может быть недопустимо. С другой стороны, при n = 500 противник, работающий в течение 200 лет, взломает систему с вероятностью около 2^{-500}. ♦

Как следует из предыдущего примера, мы можем рассматривать параметр безопасности в качестве механизма, позволяющего честным участникам «отрегулировать» безопасность системы до желаемого уровня. (С увеличением параметра безопасности увеличивается время, необходимое для работы системы, а также длина ключа. Таким образом, честные участники захотят задать настолько малый параметр безопасности, который защитит их от конкретного, вызывающего у них беспокойство класса атак.) Приравнивание параметра безопасности к длине ключа примерно сопоставимо с тем фактом, что время, необходимое для атаки методом перебора, увеличивается экспоненциально от длины ключа. Возможность «увеличения стойкости» посредством увеличения параметра безопасности имеет важные практические последствия, поскольку позволяет честным участникам защититься от роста вычислительной мощности. Следующий пример дает представление о том, как это может быть использовано на практике.

Пример 3.3

Давайте рассмотрим, какое влияние может оказать на практическую безопасность доступность более быстрых компьютеров. Допустим, у нас есть криптографическая система, в которой честные участники работают в течение $106 \cdot n2$ циклов, а противник, работая в течение $108 \cdot n^4$ циклов, может успешно «взломать» систему с максимальной вероятностью $2^{-n/2}$.

(Числа подобраны так, чтобы упростить вычисления, и не соотносятся ни с одной из существующих криптографических систем.)

Скажем, все участники используют компьютеры с процессорами с частотой 2 ГГц, а честные участники задали n = 80. Затем честные участники работают в течение $10^6 \cdot 6400$ циклов, или 3,2 секунд, а противник, работая в течение $10^8 \cdot (80)^4$ циклов, или примерно 3 недель, может взломать систему с вероятностью всего 2^{-40}.

Скажем, появляются компьютеры с процессорами с частотой 8 ГГц, и все участники обновляют компьютеры. Честные участники могут увеличить n до 160 (требуется генерация нового ключа) и работают в течение 3,2 секунд (т. е. $10^6 \cdot 160^2$ циклов при частоте $8 \cdot 10^9$ циклов в минуту). В то же время противнику теперь понадобится работать в течение более 8 миллионов секунд, или более 13 недель, чтобы достичь успеха с вероятностью 2^{-80}. Появление более быстрых компьютеров приводит к тому, что работа противники усложняется ♦

Даже при использовании асимптотического подхода важно помнить, что при разворачивании криптосистемы на практике, в конечном счете, понадобятся конкретные гарантии. (В конце концов, кто-то должен определить некоторое значение n.) Как показывают приведенные выше примеры, как правило, асимптотические заявления безопасности могут быть переведены в ограничения конкретной стойкости для любого желаемого значения n.

Асимптотический подход в деталях

Теперь мы более формально рассмотрим понятия «алгоритмы полиномиального времени» и «пренебрежительно малая вероятность успеха».

Эффективные алгоритмы . Мы определили, что алгоритм является эффективным, если работает в течение полиномиального времени. Алгоритм A работает в течение полиномиального времени, если существует многочлен p, такой что для каждого ввода $x \in \{0, 1\}^*$ вычисление A(x) оканчивается за максимум $p(|x|)$ шагов. (Здесь |x| обозначает длину строки x.) Как было сказано ранее, нас интересуют только противники, работающие в полиномиальном времени от параметра безопасности n. Поскольку мы измеряем время работы алгоритма с точки зрения длины входных данных, иногда мы предоставляем алгоритмам параметр безопасности, записанный в унарном виде (т. е. 1n или строка из n единиц), в качестве входных данных. Стороны, точнее, алгоритмы, которые они используют, помимо параметра безопасности могут использовать другие входные данные (например, сообщение, которое необходимо зашифровать), и мы допускаем, что они работают в полиномиальном времени от (общей) длины входных данных.

По умолчанию мы допускаем, что все алгоритмы являются вероятностными (или случайными). Каждый такой алгоритм может «подбрасывать монетку» на каждом этапе своей работы. Иными словами, алгоритм на каждом этапе имеет

доступ к по-настоящему произвольным битам. Аналогичным образом, мы можем рассматривать случайный алгоритм, как тот, которому помимо входных данных задана равномерно распределенная случайная лента достаточной длины1, чьи биты он может использовать в своей работе по мере необходимости. Существуют две причины, по которым мы по умолчанию рассматриваем случайные алгоритмы. Во-первых, случайность существенна для криптографии (например, при выборе случайных ключей и т.д .), поэтому честные участники должны быть вероятностными, исходя из чего естественно допускать, что и противники тоже могут быть вероятностными. Во-вторых, рандомизация практична и, насколько нам известно, наделяет перехватчиков дополнительной мощностью. Поскольку нашей целью является создание модели всех реалистичных атак, мы предпочитаем более свободное определение эффективного вычисления.

Пренебрежительно малая вероятность успеха. Пренебрежительно малая функция - это функция, которая асимптотически меньше любой обратной полиномиальной функции. Формально:

ОПРЕДЕЛЕНИЕ 3.4 *Функция f от натуральных чисел до неотрицательных действительных чисел пренебрежительно мала, если для каждого положительного многочлена p существует такое N, что для всех целых чисел n > N верно f (n) < 1/p(n).*

Для краткости вышесказанное можно также записать следующим образом: для каждого многочлена p и всех значительно больших значений n верно $f(n) < 1/p(n)$. Равноценная формулировка вышесказанного требует того, что для всех постоянных c существовало такое N, что для всех $n > N$ было верно $f(n) < n^{-c}$. Обычно мы обозначаем произвольную пренебрежительно малую функцию с помощью negl (сокращение от англ. negligible).

Пример 3.5

Функции 2^{-n}, $2^{-\sqrt{n}}$ и $n^{-\log n}$ являются пренебрежительно малыми. Однако они стремятся к нулю по-разному. Например, мы можем рассмотреть наименьшее значение n, для которого значение каждой функции будет меньше $1/n^5$:

1. Решая $2^{-n} < n^{-5}$, получаем $n > 5 \log n$. Наименьшим целым значением n, для которого это верно, является $n = 23$.

2. Решая $2^{-n} < n^{-5}$, получаем $n > 25 \log^2 n$. Наименьшим целым значением n, для которого это верно, является $n \approx 3500$.

3. Решая $n^{-\log n} < n^{-5}$, мы получаем $\log n > 5$. Наименьшим целым значением n, для которого это верно, является $n = 33$.

---

[1]Если рассматриваемый алгоритм работает на p(n) шагах входных данных длиной n, тогда случайная лента длиной p(n) является достаточной, поскольку перехватчик может считывать максимум один случайный бит за один временной шаг.

Из вышесказанного у вас может сложиться впечатление, что функция $n^{-\log n}$ стремится к нулю быстрее, чем $2^{-\sqrt{n}}$. Однако это не верно: для всех $n > 65536$ верно, что $2^{-\sqrt{n}} < n^{-\log n}$. Тем не менее, это не означает, что для значений n в сотнях или тысячах, вероятность успеха противника от $n^{-\log n}$ предпочтительнее вероятности успеха противника от $2^{-\sqrt{n}}$. ♦

Техническим преимуществом работы с пренебрежительно малыми вероятностями успеха является то, что они подчиняются определенным свойствам замыкания. Далее представлено простое упражнение.

ПРЕДПОЛОЖЕНИЕ 3.6 *Пусть* negl1 и negl2 - *это пренебрежимо малые функции. Тогда,*

1. Функция negl3, *заданная с помощью* negl3(n) = negl1(n)+negl2(n), *является пренебрежимо малой.*

2. Для любого положительного многочлена p, функция negl4, *заданная с помощью* negl4(n) = p(n) • negl1(n), является пренебрежимо малой.

Вторая часть указанного выше предположения подразумевает, что если некоторое событие происходит в некотором эксперименте только с пренебрежимо малой вероятностью, тогда событие происходит с пренебрежимо малой вероятностью даже при повторении эксперимента полиномиально много раз. (Это основывается на границе объединения. См. Предположение А.7.) Например, вероятность того, что результатом n подбрасываний симметричной монеты будет «орел», пренебрежимо мала. Это означает, что если мы повторим подбрасывание n монет полиномиально много раз, вероятность того, что результатом любого из этих экспериментов будет *n* «орлов», остается пренебрежимо малой.

Следствием второй части приведенного выше предположения является то, что если функция g не является пренебрежимо малой, тогда таковой не является ни одна из функций $f(n) \stackrel{\text{def}}{=} g(n)/p(n)$ положительного многочлена p.

Асимптотическая стойкость: Краткие выводы

Каждое определение криптографической стойкости состоит из двух частей: определение того, что считается «взломом» системы, и конкретизация возможностей противника. Возможности противника могут зависеть от многих факторов (например, в случае шифрования мы допускаем атаку на основе шифртекста или подобранного открытого текста). Что же касается *вычислительных возможностей* противника, в дальнейшем мы будем создавать модель эффективного противника и, таким образом, рассматривать только те стратегии противника, которые могут быть осуществлены в вероятностно полиномиальном времени. Определения будут всегда формулироваться так, что взлом системы, происходящий с пренебрежимо малой вероятностью, не будет считаться значительным. Таким образом, любое определение криптографической стойкости

будет подчиняться следующим общим принципам:

> Система является надежной, если для каждого *вероятностного полиномиально-временного* противника A, осуществляющего атаку одного из формально заданных типов, вероятность успешного осуществления атаки противником A (если успех также формально задан) *пренебрежимо* мала.

Такое определение является *асимптотическим*, потому что существует вероятность того, что при небольших значениях n противник с большой вероятностью добьется успеха. Для более подробного рассмотрения данного вопроса расширим понятие «пренебрежимо малый» с помощью следующего утверждения:

> Система является *криптографически надежной*, если для каждого ppt противника A, осуществляющего атаку некоторым формально заданным способом, и для каждого положительного многочлена p существует такое целое число N, что когда $n > N$, вероятность успешной атаки A меньше $1/p(n)$.

Обратите внимание, что никаких гарантий не дается для значений $n \leq N$.

О решениях, принятых при определении асимптотической безопасности

При определении общего понятия асимптотической безопасности нами было принято два решения: мы определили эффективные стратегии противника с помощью класса вероятностных *полиноминально-временных алгоритмов* и приравняли малые шансу на успех к *пренебрежимо малым вероятностям*. В некоторой мере, оба эти решения произвольны, и можно построить совершенно приемлемую теорию, назвав, скажем, эффективными стратегиями те, которые работают в квадратичном времени, или малой вероятностью на успех ту, что ограничена функцией 2^{-n}. Тем не менее, мы кратко подтвердим нами принятые нами (стандартные) решения.

Те, кто знаком с теорией сложности или со сложностью алгоритмов, осознают, что идея приравнивания эффективного вычисления к (вероятностным) полиномиально-временным алгоритмам характерна не только для криптографии. Достоинством использования (вероятностного) полиномиального времени в качестве показателя эффективности является тот факт, что нам не нужно точно задавать вычислительную модель, поскольку в соответствии с тезисом Чёрча-Тьюринга все «приемлемые» вычислительные системы являются полиномиально эквивалентными. Таким образом, нам не приходится указывать, используем ли мы машину Тьюринга, булеву схему или машину с произвольным доступом к памяти. Мы можем представить алгоритмы, составленные с помощью высокоуровневого псевдокода, и быть уверены, что эти алгоритмы будут работать в полиномиальном времени, как и любая их приемлемая реализация.

Другим достоинством (вероятностных) полиномиально-временных алгоритмов является то, что они соответствуют желаемым свойствам замыкания:

в частности, алгоритм, полиномиально много раз обращающийся к полиномиально-временной подпрограмме (и к тому же выполняющий только полиномиальные вычисления), сам будет работать в полиномиальном времени.

Наиболее важной характеристикой пренебрежимо малых вероятностей является свойство замыкание, рассмотренное нами в предположении 3.6(2): пренебрежимо малая функция, увеличенная в некоторое полиномиальное количество раз, остается пренебрежимо малой. В частности, это означает, что если алгоритм полиномиально много раз обращается к некоторой подпрограмме, которая каждый раз в результате такого обращения «терпит неудачу» с пренебрежимо малой вероятностью, то вероятность того, что какие-либо обращения к этой подпрограмме потерпят неудачу, остается пренебрежимо малой.

Необходимость ослаблений

Вычислительная стойкость вводит два ослабления совершенной стойкости: во-первых, криптографическая стойкость гарантируется только против эффективных противников; во-вторых, допускается малая вероятность успеха. Оба эти ослабления обязательны для создания практических систем шифрования и, в частности, для игнорирования отрицательных результатов абсолютного криптографически стойкого шифрования. Неформально обсудим, почему это именно так. Допустим, у нас есть система шифрования, где размер пространства ключа K гораздо меньше размера пространства сообщения M. (Как говорилось ранее в предыдущей главе, это значит, что система не может быть совершенно криптографически надежной.) Проводятся две атаки вне зависимости от конструкции системы шифрования:

• Задан шифртекст c, противник может расшифровать c, используя все ключи $k \in K$. Таким образом он получает перечень всех сообщения, к которым может относиться c. Поскольку этот перечень не содержит всех M ($|K| < |M|$), эта атака приводит к утечке некоторой информации о зашифрованном сообщении.

Кроме того, скажем, противник осуществляет атаку на основе открытых текстов и узнает шифртексты c1, ..., cA, соответствующие сообщениям m1,..., mA. Противник может снова попытаться расшифровать эти шифртексты со всеми возможными ключами, пока не найдет ключ k, для которого Deck (ci) = mi для всех i. Позднее задан шифртекст c зашифрованного неизвестного сообщения m, почти наверняка окажется, что Deck (c) = m.

Атаки методом перебора, как указанные выше, позволяют противнику добиться успеха преимущественно один раз линейно от |K|.

• Снова рассмотрим случай, когда противник узнает шифртексты c1, ..., cA, соответствующие сообщениям m1, ..., mA. Противник может угадать унифицированный ключ $k \in K$ и проверить, верно ли Deck (ci) = mi для всех i. Если все так, как указано выше, то противник может использовать k для расшифровки

всего, что в дальнейшем будет зашифровано честными участниками.

Здесь противник работает в главным образом постоянном времени и преуспевает с ненулевой (хотя и очень малой) вероятностью 1/|K|.

Отсюда следует, что если мы хотим зашифровать множество сообщений с использованием одного короткого ключа, криптографическая стойкость может быть достигнута только при ограничении времени работы противника (тогда у противника не будет достаточного количества времени для поиска методом перебора) и готовности допустить очень малую вероятность успеха (тогда вторая «атака» исключена).

3.2 Определение вычислительно криптостойкого шифрования

Разобрав вопрос предыдущего раздела, мы готовы представить определение вычислительной стойкости шифрования с закрытым ключом. Во-первых, мы повторно определим синтаксис шифрования с закрытым ключом: он в основном будет таким же, как и синтаксис, введенный в Главе 2, только мы теперь будем непосредственно принимать во внимание параметр безопасности n. Мы также позволим алгоритму расшифрования выводит ь сообщение об ошибке в случае, если ему будет задан недействительный шифртекст. Наконец, мы по умолчанию допустим, что пространство сообщения является множеством {0, 1}* всех двоичных строк (конечной длины).

ОПРЕДЕЛЕНИЕ 3.7 Система шифрования с закрытым ключом - *это кортеж вероятностных полиномиально-временных алгоритмов* (Gen, Enc, Dec), таких что:

1. Алгоритм генерации ключа Gen использует в качестве входных данных 1^n (т.е . унарно записанный параметр безопасности) и выводит ключ k; пишем $k \leftarrow Gen(1^n)$ (делая акцент на том, что алгоритм Gen является случайным). Допустим без ущерба для общности, что любой ключ k, выведенный Gen(1n) удовлетворяет условию $|k| \geq n$.

2. Алгоритм шифрования Enc на входе принимает ключ k и открытый текст сообщения $m \in \{0, 1\}^*$ и выводит шифртекст c. Поскольку алгоритм Enc может быть случайным, мы записываем это следующим образом $c \leftarrow Enck (m)$.

3. Алгоритм расшифрования Dec принимает на входе ключ k и шифртекст c, а на выходе выдает открытый текст m или сообщение об ошибке. Допустим, алгоритм Dec детерминирован, тогда мы пишем, что m := Deck (c) (допуская здесь, что алгоритм Dec не выдает ошибку). Обозначим общую ошибку символом $\perp$.

Необходимо, чтобы для каждого n каждый ключ k, выведенный алгоритмом Gen(1n) и каждого $m \in \{0, 1\}^*$, было верно, что Deck (Enck (m)) = m.

Если (Gen, Enc, Dec), такие что для k, выведенного $Gen(1^n)$, алгоритм Enck определен только для сообщений $m \in \{0, 1\}^{A(n)}$, тогда мы говорим, что (Gen, Enc, Dec) является системой шифрования с фиксированной длиной ключа для

сообщений длиной A(n).

Практически всегда алгоритм Gen(1^n) просто выводит в качестве ключа однородную n-битную строку. В этом случае мы будем опускать алгоритм Gen и просто определять систему шифрования с закрытым ключом с помощью пары алгоритмов (Enc, Dec).

Приведенное выше определение рассматривает системы, не сохраняющие информацию о состоянии, в которых каждое инициирование работы алгоритма Enc (и Dec) не зависит от предыдущего запуска работы. Позднее в этой главе мы попутно рассмотрим системы, сохраняющие состояние, в которых отправитель (и, возможно, получатель) должен поддерживать рабочее состоянии одновременно с запуском. Если иное явно не указано, все наши результаты предполагают шифрование и (или) шифрование без сохранения состояния.

3.2.1 Основное понятие криптографической стойкости

Начнем с наиболее основного понятия криптостойкости при шифровании с закрытым ключом: криптостойкость против атаки на основе шифртекста, где противник исследует только один шифртекст, или криптостойкость, когда данный ключ используется для шифрования только одного сообщения. Далее в главе мы рассмотрим более строгие определения криптостойкости.

Объяснение определения. Как было сказано ранее, любое определение криптостойкости состоит из двух отдельных компонентов: модели угроз (т.е. определения допускаемых возможностей противника) и цель безопасности (обычно определяется посредством описания того, что подразумевается под «взломом» системы). Начнем разбор определения с рассмотрения простейшей модели угроз, где *противник-перехватчик* исследует единственное зашифрованное сообщение. Это именно так модель угроз, которую мы рассматривали в предыдущей главе, за исключением того, что, как уже говорилось в предыдущем разделе, нас интересуют противники, обладающие ограниченными вычислительными возможностями, и, как следствие, работающие в полиномиальном времени.

И хотя мы сделали два допущения относительно возможностей противника (он только перехватывает сообщение и работает в полиномиальном времени), мы не сделали никаких допущений касательно *стратегии* противника, пытающегося расшифровать исследуемый им шифртекст. Это особенно важно для значимого понятия криптостойкости. Определение обеспечивает защиту от *любого* вычислительно ограниченного противника вне зависимости от используемого им алгоритма.

Правильное определение цели безопасности шифрования не просто, но мы уже подробно рассмотрели этот вопрос в разделе 1.4.1 и в предыдущей главе. Так или иначе, напомним, что смысл определения заключается в том, что противник не должен суметь узнать *любой частичной информации* об открытом тексте из шифртекста. *Определение семантической криптостойкости* (см. Раздел 3.2.2) точно фор-

мализует это понятие и представляет собой первое из когда-либо предложенных определений вычислительно надежного шифрования. Семантическая криптостойкость - сложно е понятие, с которым трудно работать. К счастью, существует равносильное и более простое определение, называемое *неразличимостью*.

Определение неразличимости сформулировано по образцу альтернативного определения совершенной криптостойкости (см. Определение 2.5). (Еще одна причина того, почему определение неопределенности хорошо.) Напомним, что Определение 2.5 включает в себя эксперимент $\Pr[\mathsf{PrivK}^{eav}_{\mathcal{A},\Pi}$, в котором противник А выводит два сообщения m0 и m1, а затем получает одно из этих сообщений, зашифрованное с помощью унифицированного ключа. В определении говорится, что система П надежна, если ни один противник А не сможет определить, какое из сообщений m0, m1 было зашифровано, с вероятностью, отличной от 1/2, которая равносильна вероятности того, что А правильно угадает ответ.

А теперь перейдем к эксперименту $\Pr[\mathsf{PrivK}^{eav}_{\mathcal{A},\Pi}$ почти аналогичному (за исключением некоторых технических различий, указанных ниже) тому, что мы уже проводили, однако, мы внесем два ключевых изменения в само определение:

1. Теперь мы рассматриваем только противников, работающих в *полиномиальном времени*, хотя в определении 2.5 рассматривались даже противники с неограниченным временем работы.

2. Теперь мы допускаем, что противник может определить зашифрованное сообщение с вероятностью *пренебрежимо большей, чем 1/2*.

Как уже было сказано в предыдущем разделе, упомянутые выше ослабления являются ключевыми элементами вычислительной криптостойкости.

Что же касается других отличий, основным является то, что теперь мы параметризуем эксперимент посредством параметра безопасности n. Тогда мы измеряем как время работы противника А, так и вероятность его успеха в качестве функций по n. Запишем $\Pr[\mathsf{PrivK}^{eav}_{\mathcal{A},\Pi}(n)$, чтобы обозначить, что эксперимент проводится с учетом параметра безопасности n, и запишем

$$\Pr[\mathsf{PrivK}^{eav}_{\mathcal{A},\Pi}(n) = 1] \qquad (3.1)$$

для обозначения вероятности того, что результатом эксперимента Pr $\Pr[\mathsf{PrivK}^{eav}_{\mathcal{A},\Pi}$ (n) является 1. Обратите внимание, что при фиксированных А, П, уравнение (3.1) является функцией по n.

Второе отличие эксперимента $\Pr[\mathsf{PrivK}^{eav}_{\mathcal{A},\Pi}$ заключается в том, что теперь нам явно требуется, чтобы противник вывел два сообщения m0, m1 *равной длины*. (В определении 2.5 это требование подразумевается, если пространство сообщения М содержит только сообщения фиксированной длины, как это происходит при шифровании Вернама.) Это означает, что по умолчанию для надежного

шифрования на не требуется скрывать длину открытого текста. Мы вернемся к этому моменту в конце данного раздела (также см. упражнения 3.2 и 3.3).

Неразличимость в присутствии подслушивающей стороны. Теперь мы дадим формальное определение, начав с описанного выше эксперимента. Эксперимент определен для любой системы шифрования с закрытым ключом $\Pi = (Gen, Enc, Dec)$, любого противника A и любого значения параметра безопасности n:

Эксперимент по неразличимости для противника $\Pr[\mathsf{PrivK}^{eav}_{\mathcal{A},\Pi}(n)]$:

1. Противнику A даны вводные данные 1^n, и он выводит пару сообщений m0, m1 при $|m0| = |m1|$.

2. С помощью алгоритма $Gen(1^n)$ генерируется ключ k и выбирается единообразный бит $b \in \{0, 1\}$. Вычисляется шифртекст $c \leftarrow Enc_k(m_b)$, который передается A. Мы рассматриваем c в качестве анализируемого шифртекста.

3. A выводит бит b^r.

4. Результатом эксперимента является 1, если $b^r = b$, иначе 0. Если $PrivK^{eav}(n) = 1$, мы говорим, что A добился успеха.

Ограничения по длине m0 и m1 отсутствуют, поскольку они равны. (Разумеется, если A работает в полиномиальном времени, тогда m0 и m1 имеют полиномиальную длину по n.) Если Π является системой фиксированной длины для сообщений длиной A(n), тогда в вышеупомянутый эксперимент добавляется требование $m0, m1 \in \{0, 1\}^{A(n)}$.

Подразумевается, что противник может только перехватывать шифртекст, поскольку его вводные данные ограничены (единственным) шифртекстом, и противник не может в дальнейшем взаимодействовать с отправителем или получателем. (Как мы увидим дальше, возможность дополнительного взаимодействия делает противника значительно сильнее.)

В соответствии с определением неразличимости, система шифрования является криптографически надежной, если ни один ppt противник A не может успешно угадать, какое сообщение зашифровано в вышеупомянутом эксперименте с вероятностью значительно превышающей случайную догадку, верную с вероятностью 1/2:

ОПРЕДЕЛЕНИЕ 3.8 *Система шифрования с закрытым ключом* $\Pi = (Gen, Enc, Dec)$ обладает неразличимым шифрованием при наличии подслушивающей стороны, или является криптографически надежной с точки зрения EAV *(расширенной проверки приложения), если для всех вероятностных полиномиально-временных противников A существует такая пренебрежимо малая функция* negl, *что для всех n*,

$$\Pr[\mathsf{PrivK}^{eav}_{\mathcal{A},\Pi}(n) = 1] \leq 1/2 + negl(n),$$

где вероятность вычисляется по случайности, используемой A, и случайности, используемой в эксперименте (для выбора ключа и бита b, а также любая случайность, используемая алгоритмом Enc).

Примечание: если не указано иное, когда мы пишем “f (n) ≤ g(n)”, это означает, что неравенство верно для всех n.

Должно быть очевидно, что определение 3.8 слабее определения 2.5, которое равносильно совершенной стойкости. Таким образом, любая совершенно криптостойкая системы шифрования обладает неразличимым шифрованием при наличии подслушивающей стороны. Отсюда следует, что наша цель - показать, что существуют системы шифрования, соответствующие вышесказанному, в которых используется ключ короче сообщения. Иными словами, мы покажем системы, которые соответствуют определению 3.8, но не могут соответствовать определению 2.5.

Равносильная формулировка. Определение 3.8 требует, чтобы ни один ppt противник не мог определить, какое из двух сообщений зашифровано, с вероятностью значительно больше 1/2. Равносильная формулировка: каждый ppt противник действует одинаково, когда видит зашифрованное сообщение m0 или m1. Поскольку A выводит один бит, «действовать одинаково» означает, что он выводит 1 в каждом случае почти с одинаковой вероятностью. Чтобы это формализовать, определим $\Pr[\mathsf{PrivK}^{eav}_{\mathcal{A},\Pi}(n, b)]$ как

указано выше за исключением того, что используется фиксированный бит b (а не выбранный случайным образом). Пусть out ($\Pr[\mathsf{PrivK}^{eav}_{\mathcal{A},\Pi}(n, b)]$) означает выведенный во время эксперимента противником A бит br. Следующее главным образом указывает на то. что ни один A не может определить, участвует он в эксперименте $\Pr[\mathsf{PrivK}^{eav}_{\mathcal{A},\Pi}(n, 0)]$ или в эксперименте $\Pr[\mathsf{PrivK}^{eav}_{\mathcal{A},\Pi}(n, 1)]$.

ОПРЕДЕЛЕНИЕ 3.9 *Система шифрования с закрытым ключом* Π = (Gen, Enc, Dec) обладает неразличимым шифрованием при наличии подслушивающей стороны, *если для всех ppt противников A существует такая пренебрежимо малая функция negl, что*

$\Pr[\text{out}_A(\Pr[\mathsf{PrivK}^{eav}_{\mathcal{A},\Pi}(n, 0))] = 1] - \Pr[\text{out}_A(\Pr[\mathsf{PrivK}^{eav}_{\mathcal{A},\Pi}(n, 1))] = 1]. \leq \text{negl}(n)$.

Доказательство равносильности данного определения определению 3.8 оставлено в качестве упражнения.

Шифрование и длина открытого текста

Базовое понятие надежного шифрования не требует того, чтобы системы шифрования скрывали длин открытого текста и, фактически, все широко используемые системы шифрования обнаруживают длину открытого текста (или близкую к ней). Основная причина в том, что невозможно поддерживать сообщения произвольной длины при сокрытии всей информации об открытом тексте (см. Упражнение 3.2). Во многих случаях это несущественно, поскольку открытый текст либо уже не секретен, либо не важен. Однако так происходит не всегда, иногда утечка информации о длине открытого текста представляет проблему. Например:

• *Простые численные и (или) текстовые данные:* Скажем, используемая система шифрования точно воспроизводит длину открытого текста. Тогда зашифрованная информация о заработной плате раскроет, что некто получает 5-значную или 6-значную заработную плату. Точно также шифрование ответов «да»/»нет» позволит точно догадаться об ответе.

• *Автозаполнение:* Веб-сайты часто имеют функцию «автозаполнения», или «выпадающего списка», по которому сервер предлагает список возможных слов или фраз, основываясь на частичной информации, которую пользователь уже ввел. Размер этого списка может раскрыть информацию о буквах, которые пользователь уже ввел. (Например, число автозаполнений для «th» гораздо больше, чем для «zo».)

• *Поиск в базе данных:* Рассмотрим случай, когда пользователь создает запрос обо всех записях базы данных, соответствующих некоторому критерию поиска. Количество выданных записей может раскрыть много информации о том, что искал пользователь. Это особенно может принести ущерб, если пользователь ищет медицинские данные и запрос раскрывает информацию о заболевании пользователя.

• *Сжатые данные*: Если открытый текст был сжат перед шифрованием, тогда информация об открытом тексте может быть раскрыта даже при шифровании данных только фиксированной длины. (Вследствие этого такая система шифрования не будет соответствовать определению 3.8.) Например, короткий сжатый открытый текст будет указывать на то, что изначальный (несжатый) открытый текст обладает избыточностью. Если противник может управлять частью того, что было зашифровано, такая уязвимость может привести к тому, что противник узнает дополнительную информацию об открытом тексте. Доказано возможное использование атаки такого рода (ПРЕСТУПНАЯ атака) против зашифрованного HTTP трафика для раскрытия куки (cookie) закрытой сессии.

При использовании шифрования необходимо определить, является ли утечка информации о длине открытого текста критичной, и если это так, необходимо принять меры для смягчения или предотвращения такой утечки, перед шифрованием необходимо дополнить сообщение битами заполнителя до указанной длины.

3.2.2 *Семантическая криптостойкость

Мы пояснили определение надежного шифрования, установив, что противник не должен суметь получить частичную информацию об открытом тексте из шифртекста. Однако определение неразличимости выглядит совершенно иначе. Как было сказано ранее, определение 3.8 равносильно определению семантической криптостойкости, формализующему понятие того, что частичная информация не должна быть раскрыта. Мы пришли к тому определению в ходе обсуждения двух более слабых понятий и доказав, что они подразумеваются неразличимостью.

Мы начнем с того, что покажем, что неразличимость означает, что шифртекст не допускает утечки никакой информации об отдельных битах открытого текста. Формально скажем, что система шифрования (Enc, Dec) является надежной с точки зрения EAV (тогда вспомним, когда алгоритм Gen опускается, ключ представляет собой однородную n-битную строку), а m $\in$ $\{0, 1\}^A$ однородным. Тогда докажем, что для любого индекса i, невозможно угадать mi по Enck (m) (где, в этом разделе, m^i обозначает i-тый бит m) с вероятностью больше, чем 1/2.

ТЕОРЕМА 3.10 *Пусть* П = (Enc, Dec) - *это система шифрования с закрытым ключом фиксированной длины A, которая обладает неразличимым шифрованием при наличии подслушивающей стороны. Тогда для всех ppt противников A и любых i* $\in$ {1, ..., A}, *существует пренебрежимо малая функция* negl, *такая что*1

$$\Pr\left[\mathcal{A}(1^n, \mathsf{Enc}_k(m)) = m^i\right] \leq \frac{1}{2} + \mathsf{negl}(n),$$

где вероятность вычисляется по однородным *m* $\in$ *{0, 1}^{A} и k* $\in$ *{0, 1}^{n}, произвольному* A и *произвольному* Enc.

ДОКАЗАТЕЛЬСТВО Смысл доказательства данной теоремы заключается в том, что если возможно угадать i-тый бит m по Enck (m), тогда также возможно различить шифрования двух сообщений m0 и m1, чьи i-тые биты различаются. Формализуем это используя доказательство от противного, в котором мы покажем, как использовать любого эффективного противника A для создания такого эффективного противника A^r, что если A нарушает понятие безопасности по теореме для П, тогда A^r нарушает определение неразличимости для П. (См. Раздел 3.3.2.) Поскольку П обладает неразличимым шифрованием, то она также должна быть надежной по теореме.

Зафиксируем произвольного ppt противника A и i $\in$ {1, ..., A}. Пусть I0 $\in${0, 1}^{A} - это множество всех строк, чей i-тый бит равен 0, и пусть I1 $\in$ $\{0, 1\}^A$ - это множество всех строк, чей i-тый бит равен 1. Мы имеем

$$\Pr\left[\mathcal{A}(1^n, \mathsf{Enc}_k(m)) = m^i\right]$$
$$= \frac{1}{2} \cdot \Pr_{m_0 \leftarrow I_0}\left[\mathcal{A}(1^n, \mathsf{Enc}_k(m_0)) = 0\right] + \frac{1}{2} \cdot \Pr_{m_1 \leftarrow I_1}\left[\mathcal{A}(1^n, \mathsf{Enc}_k(m_1)) = 1\right].$$

Сконструируем следующего противника-перехватчика A^r:

Противник A^r:

1. Выбирает однородные m0 $\in$ I0 и m1 $\in$ I1. Выводит m0, m1.

2. После изучения шифртекста c вызывает A(1^n, c). Если A выводит 0, выводить $b^r = 0$; иначе, выводить $b^r = 1$.

A^r работает в полиномиальном времени, так как A работает в полиномиальном времени.

По определению эксперимента $\Pr\left[\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\Pi}\right.$ (n) имеем, что A^r добивается успеха тогда и только тогда, когда A выводит b при получении Enck (mb). Таким образом,

$$
\begin{aligned}
&\Pr\left[\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A}',\Pi}(n)=1\right]\\
&=\Pr\left[\mathcal{A}(1^n,\mathsf{Enc}_k(m_b))=b\right]\\
&=\frac{1}{2}\cdot\Pr_{m_0\leftarrow I_0}\left[\mathcal{A}(1^n,\mathsf{Enc}_k(m_0))=0\right]+\frac{1}{2}\cdot\Pr_{m_1\leftarrow I_1}\left[\mathcal{A}(1^n,\mathsf{Enc}_k(m_1))=1\right]\\
&=\Pr\left[\mathcal{A}(1^n,\mathsf{Enc}_k(m))=m^i\right].
\end{aligned}
$$

По гипотезе о том, что (Enc, Dec) обладает неразличимым шифрованием при наличии подслушивающей стороны, то существует пренебрежимо малая функция negl, такая что $\Pr\left[\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n)=1\right] \leq 1 + \mathsf{negl}(n)$. Мы приходим к выводу, что

$$\Pr\left[\mathcal{A}(1^n,\mathsf{Enc}_k(m))=m^i\right]\leq\frac{1}{2}+\mathsf{negl}(n),$$

завершая доказательство.

Затем мы заявляем, что неразличимость, грубо говоря, означает, что ни один ppt перехватчик не может узнать *никакой* функции открытого текста из данного шифртекста вне зависимости от распределения отправляемых сообщений. Это предполагает мысль о том, что никакая информация об открытом тексте не просочится из получившегося шифртекста. Однако это требование не так просто определить формально. Чтобы это понять, обратите внимание на то, что даже в вышеупомянутом случае, легко вычислить i-тый бит m, если m равномерно выбирается из, скажем, множества всех строк, чей i-тый бит равен 0 (по сравнению с равномерно выбранным из $\{0,1\}^A$). Таким образом, мы хотим сказать, что если существует противник, который с некоторой вероятностью правильно вычисляет f (m), когда задан Enck (m), тогда существует противник, который может с той же вероятностью правильно вычислить f (m), не имея заданного шифртекста совсем (а только зная распределение m). В дальнейшем мы сосредоточимся на случаях, когда m равномерно выбирается из множества $S \subseteq \{0,1\}^A$.

ТЕОРЕМА 3.11 *Пусть* (Enc, Dec) - *это система шифрования с закрытым ключом фиксированной длины A, которая обладает неразличимым шифрованием при наличии подслушивающей стороны. Тогда для любого ppt алгоритма A существует ppt алгоритм* A^r, *такой что для любого* $S \subseteq \{0,1\}^A$ и любой функции $f: \{0,1\}^A \rightarrow \{0,1\}$, существует пренебрежимо малая функция negl, такая что:

$$\left|\Pr\left[\mathcal{A}(1^n,\mathsf{Enc}_k(m))=f(m)\right]-\Pr\left[\mathcal{A}'(1^n)=f(m)\right]\right|\leq\mathsf{negl}(n),$$

где первая вероятность вычисляется по равномерному выбору $k \in \{0,1\}^n$ и $m \in S$, произвольному A и произвольному Enc, а вторая вероятность вычисляется по равномерному выбору $m \in S$ и произвольному A^r.

ДОКАЗАТЕЛЬСТВО (набросок) Тот факт, что (Enc, Dec) является надежным с точки зрения EAV означает, что для любого $S \subseteq \{0,1\}^A$ не существует ppt противника, который мог бы отличить Enck (m) (для однородного $m \in S$) и Enck (1^A). Теперь рассмотрим вероятность того, что A успешно вычислит f

(m), получив Enck (m). Мы заявляем, что A должен успешно вычислить f (m), получив Enck (1^A), с примерно той же вероятностью; иначе, A может быть использован, чтобы различать Enck (m) и Enck (1^A). Дистинктор легко сконструирован: равномерно выбираем m ∈ S и выводим m0 = m, m1 = 1^A . После получения шифртекста c, представляющего собой зашифрованное m0 или m1, вызываем A(1^n, c) и выводим 0 тогда и только тогда, когда A выводит f (m). Если A выводит f (m) после получения шифрования m с вероятностью, значительно отличающейся от вероятности вывода f (m) после получения шифрования 1^A, тогда описанный дистинктор нарушает определение 3.9.

Вышесказанное предполагает, что алгоритм A^r, не получающий c = Enck(m), еще может вычислять f (m) практически также, как это делает A: A^r(1n) выбирает унифицированный ключ k ∈ {0, 1}n, вызывает A по c ← Enck (1^A) и выводит все то же, что и A. Из вышесказанного имеем, что A выводит f (m), когда он работает как подпрограмма A^r с практически той же вероятностью, как когда он получает Enck (m). Таким образом, A^r соответствует необходимым заявленным свойствам.

Семантическая криптостойкость. Полное определение семантической криптостойкости гарантирует значительно больше, чем свойства в Теореме 3.11. Определение допускает, что длина открытого текста зависит от параметра безопасности, а также допускает весьма произвольное распределение по открытым текстам. (Фактически, мы допускаем только *эффективно отбираемые* распределения. Это значит, что существует некоторый вероятностно полиномиально-временной алгоритм Samp, такой что Samp(1^n) выводит сообщения в соответствии с распределением.) Определение также учитывает «внешнюю» информацию h(m) об открытом тексте, которая может просочиться к противнику другими способами (например, одно и то же сообщение m используется также для других целей).

ОПРЕДЕЛЕНИЕ 3.12 *Система шифрования с закрытым ключом (Enc, Dec) является семантически надежной при наличии подслушивающей стороны, если для каждого ppt алгоритма A существует такой ppt алгоритм Ar, что для любого ppt алгоритма Samp и вычислимых полиномиально-временных функций f и h следующее является пренебрежимо малым*:

$$\left|\Pr[\mathcal{A}(1^n, \mathsf{Enc}_k(m), h(m)) = f(m)] - \Pr[\mathcal{A}'(1^n, |m|, h(m)) = f(m)]\right|$$

где первая вероятность вычисляется по равномерному k ∈ {0, 1}n, m, выведенному алгоритмом Samp(1^n), произвольному A и произвольному Enc, а вторая вероятность вычисляется по m, выведенному Samp(1^n), и произвольному A^r .

Противнику A дан шифртекст Enck (m), а также внешняя информация h(m). Он пытается угадать значение f (m). Алгоритм A^r также пытается угадать значение f (m), но ему заданы только функция h(m) и длина сообщения m. В соответствии с требованиями надежности вероятность того, что A правильно

угадает f (m), должна быть примерно равна вероятности правильной догадки, сделанной A^r. Тогда, интуитивно, шифртекст Enck(m) не раскрывает никакой дополнительной информации о значении f (m).

Определение 3.12 основывается на очень сильной и убедительной формулировке гарантий безопасности, которые должна обеспечивать система шифрования. Однако работать с определением неразличимости (определение 3.8) проще. К счастью, определения являются равносильными:

ТЕОРЕМА 3.13 *Система шифрования с закрытым ключом обладает неразличимостью при наличии подслушивающей стороны тогда и только тогда, когда она является семантически надежной при наличии подслушивающей стороны.*

Забегая вперед, скажем, что подобная равносильность между семантической безопасностью и неразличимостью известна для всех определений, представленных как в данной главе, так и в главе 11. Вследствие этого, мы можем использовать определение неразличимости в качестве рабочего, пока мы уверены в том, что достигнутые гарантии соответствуют семантической надежности.

3.3 Создание криптостойких систем шифрования

Определив, какая система шифрования является надежной, читатель, вероятно, ждет, что мы немедленно перейдем к конструированию криптографически надежных систем шифрования. Однако перед этим нам необходимо ввести понятия «*псевдослучайные генераторы*» (ПСГ) и «*поточные шифры*», являющиеся важными структурными элементами шифрования с закрытым ключом. Они в свою очередь подведут нас к обсуждению *псевдослучайности*, которая играет важнейшую роль для криптографии в целом и для шифрования с закрытым ключом в частности.

3.3.1 Псевдослучайные генераторы и поточные шифры

Генератор псевдослучайных чисел *tt* - это эффективный, детерминированный алгоритм для преобразования коротких, однородных строк, так называемых *начальных чисел*, в более длинные, «выглядящие однородными» (или «псевдослучайные») выходные строки. Иначе говоря, генератор псевдослучайных чисел использует небольшой объем реальной случайности для генерации большого объема псевдослучайности. Это может быть полезно всякий раз, когда требуется большой объем случайных (выглядящих случайными) битов, поскольку по-настоящему случайные биты генерируются медленно и сложно. (См. пояснение в начале Главы 2.) Действительно, псевдослучайные генераторы изучаются с 1940-х гг., когда они были предложены для работы со статистическими моделями. Тогда исследователи предложили различные статистические испытания, которые генератор псевдослучайных чисел должен был пройти, чтобы считаться «хорошим». Например, первый бит, выведенный генератором псевдослучайных чисел, должен быть равен 1 с вероятностью, очень близкой к 1/2

(где вероятность вычисляется по равномерному выбору начальных чисел), поскольку первый бит равномерной строки равен 1 с вероятностью ровно 1/2. Действительно, соответствие любого фиксированного подмножества выведенных битов также должно равняться 1 с вероятностью, очень близкой 1/2. Также могут быть рассмотрены и более сложные статистические испытания.

Такой исторический подход определения качества потенциально пригодного генератора псевдослучайных чисел является ситуативным, и не всегда очевидным, прохождение некоторого набора статистических тестов достаточно для обеспечения надежности использования потенциально пригодного генератора псевдослучайных чисел для некоторого приложения. (В частности, может существовать другой статистический тест, который *успешно отличит* выходные данные генератора от по-настоящему случайных битов.) Исторический подход представляет еще большую проблему при использовании генераторов псевдослучайных чисел для криптографического применения. В таких условиях криптостойкость системы может быть скомпрометирована, если перехватчик сумеет отличить равномерные и выходные данные генератора; и мы не можем заранее определить, какую стратегию выберет перехватчик.

Приведенные выше соображения объясняют криптографический подход к определению генераторов псевдослучайных чисел в 1980-х гг. Основным было то, что хороший генератор псевдослучайных чисел должен пройти все (эффективные) статистические испытания. Иными словами, для любого эффективного статистического испытания (или *дистинктора*) D, вероятность того, что D выдаст 1 при получении выходных данных генератора псевдослучайных чисел, должна быть близка к вероятности того, что D выдаст 1 при получении однородной строки той же длины. Тогда, неформально, любому эффективному наблюдателю выходные данные генератора псевдослучайных чисел должны «казаться» однородной строкой.

(Обращаем ваше внимание на то, что, формально говоря, не имеет смысла говорить о том, что любая фиксированная строка является «псевдослучайно», также как и бессмысленно называть любую строку «случайной». Точнее, псевдослучайность - это свойство распределения по строкам. Тем не менее, мы иногда неформально называем строку, отобранную в соответствии с равномерным распределением, «однородной строкой», а строку, выданную генератором псевдослучайных чисел - «псевдослучайно строкой».)

Мы получаем другое представление, когда определяем, что значит псевдослучайное распределение. Пусть Dist является распределением по A-битным строкам. (Это значит, что Dist присваивает некоторую вероятность каждой строке в $\{0, 1\}^A$. Выборка из Dist означает, что мы выбираем A-битную строку в соответствии с этим распределением вероятности.) Неформально, Dist является *псевдослучайным*, если эксперимент, в котором строка отбирается из Dist,

неразличим с экспериментом, в котором отбирается однородная строка с длиной A. (Строго говоря, поскольку мы находимся в асимптотическом окружении, нам необходимо говорить о псевдослучайности последовательности распределений Dist = {Distn}, где распределение Distn используется в качестве параметра безопасности n. В настоящем обсуждении мы этот момент игнорируем.) Точнее, любой полиномиально-временной алгоритм не должен быть в состоянии определить (с вероятностью выше, чем при догадке), дана ли ему строка, отобранная в соответствии с Dist, или же ему дана однородная A-битная строка. Это означает, что *псевдослучайная строка так же хороша, как и однородная строка*, пока мы рассматриваем только полиномиально-временных наблюдателей. Как и неразличимость, являющаяся вычислительным ослаблением абсолютная стойкость , псевдослучайность является вычислительным ослаблением истинной случайности. (Мы обобщим это представление, когда будем обсуждать понятие неразличимости в главе 7.)

Теперь пусть $tt : \{0,1\}^n \rightarrow \{0,1\}^A$ является функцией, a Dist - распределением по A-битным строкам, полученным в результате выбора однородного $s \in \{0,1\}^n$ и вывода tt(s). Тогда tt является генератором псевдослучайных чисел тогда и только тогда, когда распределение Dist является псевдослучайным.

Формальное определение. Как уже говорилось выше, tt является генератором псевдослучайных чисел, если ни один эффективный дистинктор не может определить, задана ли ему строка, выданная tt, или строка, равномерно выбранная случайным образом. Как и в определении 3.9, это формализовано требованием того, что каждый эффективный алгоритм выводит 1 практически с той же вероятностью, когда задана tt(s) (для равномерного начального числа s) или равномерная строка. (За равносильным определением, аналогичным определению 3.8, обратитесь к упражнению 3.5.) Мы получаем определение в асимптотическом окружении, допуская, что параметр безопасности n определяет длину начального числа. Затем мы настаиваем на том, что tt может быть вычислен эффективным алгоритмом. Формально, нам также необходимо, чтобы выходные данные tt были длиннее входных, иначе tt не является полезным или интересным.

ОПРЕДЕЛЕНИЕ 3.14 *Пусть A - многочлен, и пусть tt - это детерминированный полиномиально-временной алгоритм, такой что для любого n и любого введенного $s \in \{0,1\}^n$, результатом tt(s) является строка длиной A(n). Мы говорим, что tt – псевдослучайный генератор, если выполняются следующие условия:*

1. (Расширение:) Для каждого n справедливо, что A(n) > n.

2. (Псевдослучайность:) Для любого ppt алгоритма D существует пренебрежимо малая функция negl, такая что

$$\left|\Pr[D(G(s)) = 1] - \Pr[D(r) = 1]\right| \leq \mathsf{negl}(n),$$

где первая вероятность вычисляется над равномерно выбранными $s \in \{0, 1\}^n$ и произвольным D, а вторая вероятность вычисляется по равномерно выбранным $r \in \{0, 1\}^{A(n)}$ и произвольным D.

Мы называем A коэффициентом расширения tt.

Приведем пример небезопасного генератора псевдослучайных чисел, чтобы ознакомиться с определением.

Пример 3.15

Определим tt(s) выводить s вместе с $\oplus_{i=1}^{n} s_i$, таким образом, коэффициент расширения tt составляет A(n) = n + 1. Выходные данные tt можно легко отличить от однородных.

Рассмотрим следующий эффективный дистинктор D: на входе строка w, на выходе 1 тогда и только тогда, когда конечный бит w равносилен операции исключающего ИЛИ всех предшествующих битов w. Поскольку эта характеристика верна для всех строк, выведенных tt, имеем Pr[D(tt(s)) = 1] = 1. С другой стороны, если *w* является равномерной, последний бит *w* является равномерным, и тогда Pr[D(w)=1]=1/2. Количество $|1/2 - 1|$ является постоянным, не пренебрежимо малым, и, таким образом, данный tt не является генератором псевдослучайных чисел. (Обратите внимание, что *D* не всегда «корректен», поскольку иногда выводит 1 даже когда задана равномерная строка. Это не меняет того факта, что *D* является хорошим дистинктором.) ♦

Обсуждение. Распределение выходных данных генератора псевдослучайных чисел tt далеко от равномерного. Рассмотрим для наглядности случай, когда $A(n)=2n$ и тогда *tt* удваивает длин входных данных. При равномерном распределении по $\{0, 1\}^{2n}$, каждая из 2^{2n} возможных строк выбирается с вероятностью ровно 2^{-2n}. Для сравнения рассмотрим распределение вывода tt (когда tt запущен по начальному числу). Когда tt получает входные данные длиной n, число различных строк в диапазоне tt составляет максимум 2n. Относительное количество строк длиной 2n, находящихся в диапазоне tt, таким образом, составляет максимум $2^n/2^{2n} = 2^{-n}$, и мы видим, что подавляющая часть строк длиной 2^n не встречаются в виде выходных данных tt.

В частности, это означает, что легко отличить случайные строки от псевдослучайных, *заданных неограниченное число раз*. Пусть tt останется тем, что указано выше, и рассмотрим экспоненциально-временной дистинктор D, работающий следующим образом: D(w) выводит 1 тогда и только тогда, когда существует $s \in \{0, 1\}^n$, такое что tt(s) = w. (Это вычисление осуществляется в экспоненциальном времени путем тщательной обработки tt(s) для каждого $s \in \{0, 1\}^n$. Напомним, что по принципу Керкгоффса характеристика tt известна D.) Теперь, если w была выведена tt, тогда D выводит 1 с вероятностью 1. Напротив, если w равномерно распределена по $\{0, 1\}^{2n}$, тогда вероятность существо-

вания s при tt(s) = w составляет максимум 2^{-n}, и тогда D выводит в этом случае 1 с вероятностью максимум 2−n. Таким образом,

$$\Pr[D(r) = 1] - \Pr[D(tt(s)) = 1]. \geq 1 - 2^{-n},$$

являющееся большим. Это еще один пример *атаки методом перебора*, который не противоречит псевдослучайности tt, поскольку атака не эффективна.

Начальное число и его длина. Начальное число для генератора псевдослучайных чисел сродни криптографическому ключу системы шифрования; начальное число должно выбираться равномерно и храниться в секрете от любого противника. Еще один важный момент, очевидный из обсуждения атак методом перебора, заключается в том, что s должно быть достаточно длинным, чтобы невозможно было перечислить все возможные начальные числа. В асимптотическом смысле это предусматривается посредством задания длины начального числа равного параметру безопасности, чтобы поиск методом полного перебора всех возможных начальных чисел требовал экспоненциального времени. На практике же начальное число должно быть достаточно длинным, чтобы было невозможно перепробовать все возможные начальные числа в пределах некоторых временных границ.

О существовании генераторов псевдослучайных чисел. Существуют генераторы псевдослучайных чисел? Безусловно, кажется, что создать их достаточно сложно, и кто-то справедливо может задаваться вопросом, существует ли какой-либо алгоритм, соответствующий определению 3.14. Хотя мы не знаем, как безоговорочно доказать существование генераторов псевдослучайных последовательностей, у нас есть веские основания верить, что они существуют. Например, они могут быть созданы при достаточно слабом допущении, что существуют *необратимые функции* (что верно, если некоторые задачи, как разложение больших чисел на множители, сложны). Мы подробнее обсудим это в Г лаве 7. Также мы имеем несколько практических конструкций, потенциально возможных генераторов псевдослучайных последовательностей, называемых *поточными шифрами*, для которых неизвестны эффективные дистинкторы. (Позднее мы представим более сильные примитивы, называемые *блочными шифрами.*) Далее мы сделаем обобщенный обзор поточных шифров и обсудим конкретные поточные шифры в Г лаве 6.

Поточные шифры

Наше определение генератора псевдослучайных последовательностей ограничено двумя способами: коэффициент расширения зафиксирован, и генератор производит полные выходные данные «за один раз». Поточные шифры, на практике используемые для реализации работы псевдослучайных генераторов, работают несколько иначе. Выходные псевдослучайные биты поточного шифра производятся постепенно и по требованию, чтобы приложение могло запросить столько псевдослучайных битов, сколько необходимо. Таким образом повышается эф-

фективность (приложение может запросить меньше битов, если их достаточно) и гибкость (отсутствует верхняя граница числа запрашиваемых битов).

Формально мы рассматриваем поточный шифр[2] как пару детерминированных алгоритмов (Init, GetBits), где:

• Init использует в качестве входных данных начальное число s и необязательный *вектор инициализации* IV и выводит исходное состояние st0.

• GetBits использует в качестве входных данных информацию о состоянии sti и выводит бит y и обновленное состояние sti+1. (Фактически, y - это блок из нескольких битов; здесь для простоты и обобщенности мы рассматриваем y как отдельный бит.)

Получив поточный шифр и любой желаемый коэффициент расширения A, мы можем определить алгоритм ttA как преобразующий входные данные длиной n в выходные данные длиной A(n). Алгоритм просто запускает Init, а затем повторно запускает GetBits в общей сложности A раз.

АЛГОРИТМ 3.16
Конструирование ttÆ из (Init, GetBits)
Вход: Начальное число s и необязательный вектор инициализации IV
Выход: y1, . . . , yÆ
st0 := Init(s, IV)
для i = 1 до A:
(yi, sti) := GetBits(st_{i-1})

Поточный шифр является *криптографически надежным* в общем смысле, если он не использует IV и для любого полиномиального A при A(n) > n сконструированная выше функция ttA является псевдослучайным генератором с коэффициентом расширения A. Мы коротко обсудим одно возможно понятие криптостойкости поточного шифра, использующего IV в разделе 3.6.1.

3.3.2 Доказательства от противного

Если мы хотим доказать, что данная конструкция вычисляемо надежная, тогда мы должны опираться на недоказанные допущения3 (в случае если система не является информационно-теоретически надежной). Сначала мы предположим, что некоторая математическая задача является сложной или что некоторый криптографический примитив *низкого уровня* надежен, а затем *докажем*,

---

[2] Здесь используется не совсем стандартная терминология, обратите внимание на то, что понятие «поточный шифр» используется разными людьми по-разному (хотя и в связанных между собой случаях). Например, некоторые используют его, когда говорят о ttÆ (см. ниже), а некоторые используют его, когда говорят о конструкции 3.17 в качестве примера ttÆ .

что данная конструкция, опирающаяся на эту задачу и (или) этот примитив, надежна при этом допущении. В разделе 1.4.2 мы уже очень подробно объяснили, почему такой подход предпочтителен, поэтому мы больше не будем повторять свои аргументы.

Доказательство того, что криптографическая конструкция надежна, пока некоторая задача трудна для решения, обычно сопровождается явно выраженным *доказательством от противного*, показывающим, как преобразовать любого эффективного противника A, который успешно «взломает» систему, в эффективный алгоритм Ar, который решит задачу, считающуюся сложной для решения. Поскольку это так важно, мы пройдемся по общей схеме такого доказательства более детально. (Мы рассмотрим множество конкретных примеров в книге, начиная с доказательства теоремы 3.18.) Начнем с предположения о том, что некоторая задача X не может быть решена (в некотором точно определенном смысле) с помощью полиномиально-временного алгоритма за исключением пренебрежимо малой вероятности. Мы хотим доказать, что некоторая криптографическая конструкция П является надежной (опять-таки, в некотором точно определенном смысле). Доказательство осуществляется по следующим этапам (см. также Рисунок 3.1):

3.3.2.1 Зафиксируем некоторого эффективного (т.е. вероятностного полиномиально-временного) противника A, атакующего П. Определим, что вероятность успеха этого противника составляет $\varepsilon(n)$.

3.3.2.2 Сконструируем эффективный алгоритм A^r, называемый «редукция», который пытается решить задачу X, используя противника A в качестве подпрограммы. Здесь важный момент заключается в том, что A^r ничего не знает о то, как работает A. Единственное, что известно A^r: предполагается, что A атакует П. Итак, задан некоторый вводный образец x задачи X, наш алгоритм A^r будет моделировать для A такой образец П, что:

3.3.2.2.1 Поскольку A может сказать, что взаимодействует с П. То есть вид A при запуске в качестве подпрограммы A^r должен быть распределен идентично (или хотя бы близко) виду A при взаимодействии с П напрямую.

3.3.2.2.2 Если A успешно «взламывает» образец П, симулированный с помощью A^r, это должно позволить A^r решить заданный ему образец x, хотя бы с обратной полиномиальной вероятностью $1/p(n)$.

3.3.2.3 Взятые в совокупности 2(a) и 2(b) подразумевают, что A^r решает X с вероятностью $\varepsilon(n)/p(n)$. Если $\varepsilon(n)$ не пренебрежимо мала, тогда таковой не является $\varepsilon(n)/p(n)$. Кроме того, если A эффективен, тогда мы получаем эффективный алгоритм Ar, решающий X с не пренебрежимо малой вероятностью, что противоречит первоначальному допущению.

---

[3]В частности, основным требованием криптографии является недоказанное допущение P f= N P.

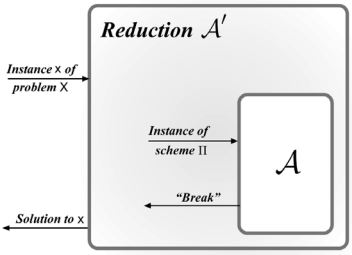

РИСУНОК 1: Общий обзор доказательства надежности от уменьшения

3.3.2.4 Принимая во внимание допущение касательно X, мы приходим к выводу, что ни один эффективный противник A не может успешно взломать П с не пренебрежимо малой вероятностью. Иными словами, П является вычисляемо надежной.

В следующем разделе мы точно проиллюстрируем вышеупомянутую мысль: мы покажем, как использовать любой псевдослучайный генератор G для конструирования системы шифрования. Мы докажем, что система шифрования является надежной, показав, что любой противник, который может «взломать» систему шифрования может различать выводные данные G и равномерные строки. Если допустить, что G является псевдослучайным генератором, тогда система шифрования является надежной.

3.3.3 Криптостойкая система шифрования фиксированной длины

Псевдослучайный генератор является естественным способом сконструировать надежную систему шифрования фиксированной длины с ключом короче сообщения. Напомним, что в шифре Вернама (см. Раздел 2.2) шифрование осуществляется с помощью операции исключающего ИЛИ в отношении случайного набора данных и сообщения. Понимание следующее: мы, на самом деле, можем использовать псевдослучайный набор данных. Вместо того, чтобы обмениваться такой долго псевдослучайной последовательностью, отправитель и получатель могу обменяться начальным числом, которое используется для генерации последовательности, когда это понадобится (см. Рисунок 3.2). Это начальное число будет короче, чем последовательно и, следовательно, короче сообщения. Что касается криптостойкости, интуиция подсказывает, что псевдослучайная строка «выглядит случайной» для любого полиномиально-временного противника и, следовательно, вычисляемо ограниченная подслушивающая сторона не может определить, зашифровано ли сообщение с помощью шифра Вернама или «псевдошифра» Вернама.

Система шифрования . Зафиксируем некоторое сообщение длиной A и допустим, что tt является псевдослучайным генератором с коэффициентов расширения A (то есть $|tt(s)| = A(|s|)$). Напомним, что система шифрования определена

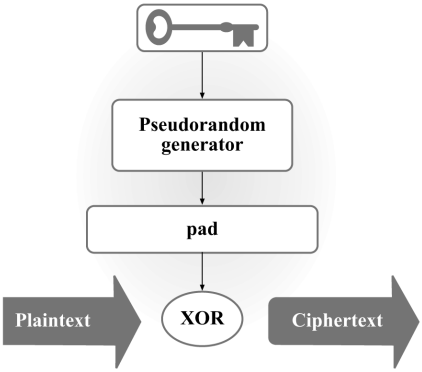

РИСУНОК 3.2: Шифрование с псевдослучайным генератором.

тремя алгоритмами: алгоритмом генерации ключей Gen, алгоритмом шифрования Enc и алгоритмом расшифрования Dec. Алгоритм генерации ключей самый обычный: Gen(1^n) просто выводит унифицированный ключ k длиной n. Шифрование осуществляется с применением tt к ключу (который служит начальным числом) с целью получения набора данных, к которым вместе с открытым текстом будет применяться операция исключающего ИЛИ. Для восстановления сообщения алгоритм расшифрования применяет tt к ключу и использует операцию XOR к получившейся последовательности и шифртексту. Система формально описана в конструкции 3.17. В разделе 3.6.1, мы описываем, как поточные шифры используются для реализации этой системы на практике.

Система шифрования с закрытым ключом, опирающаяся на любой псевдослучайный генератор.

КОНСТРУКЦИЯ 3.17

Пусть tt - псевдослучайный генератор с коэффициентом расширения A. Определим систему шифрования с закрытым ключом для сообщений длинны A следующим образом:

- Gen: на входе 1^n выбрать равномерный $k \in \{0.1\}^n$ и вывести его в качестве ключа
- Dec: на входе ключ $k \in \{0.1\}^n$ и шифротекст $m \in \{0.1\}^{A(n)}$, на выходе c: $G(k) \oplus m$
- Dec: на входе ключ $k \in \{0.1\}^n$ и шифротекст $c \in \{0.1\}^{A(n)}$, на выходе m: $G(k) \oplus c$

ТЕОРЕМА 3.18 *Если tt является псевдослучайным генератором, тогда Конструкция 3.17 является системой шифрования с закрытым ключом фиксированной длины, обладающей неразличимым шифрованием при наличии подслушивающей стороны.*

ДОКАЗАТЕЛЬСТВО Пусть Π обозначает конструкцию 3.17. Покажем, что Π соответствует определению 3.8: покажем, что для любого вероятностного полиномиально-временного противника A существует такая пренебрежимо малая функция negl, что

$$\Pr\left[\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1\right] \leq \frac{1}{2} + \mathsf{negl}(n)$$

Интуиция подсказывает, что если Π использует равномерную последовательность вместо псевдослучайной последовательности tt(k), тогда получившаяся система должна быть идентична системе шифрования Вернама, и A не должен быть в состоянии правильно угадать, какое сообщение было зашифровано, с вероятностью выше, чем 1/2. Таким образом, если уравнение (3.2) не верно, тогда A должен в неявной форме различать выходные данные tt и по-настоящему случайные строки. Мы явным образом докажем это методом от противного, именно, показав, как использовать A для конструирования эффективного дистинктора D, такой что способность D различать выходные данные tt и равномерную строку будет напрямую связана со способностью A определять, какое именно сообщение было зашифровано системой Π. Надежность tt тогда подразумевает надежность Π.

Пусть A - это произвольный ppt противник. Мы конструируем дистинктор D, который на входе берет строку w и целью которого является определить, была ли строка w выбрана равномерно (т. е. w - это «случайная строка») или w была сгенерирована с помощью унифицированного k и вычислена w := tt(k) (т. е. w - это «псевдослучайная строка»). Мы конструируем D так, чтобы имитировал эксперимент с подслушивающей стороной для A, как описано ниже, и наблюдал, добьется A успеха или нет. Если A справляется, тогда D угадывает, что w должна быть псевдослучайной строкой, а если A не справляется, тогда D угадывает, что w является случайной строкой. Более подробно:

Дистинктор D:

На входе D задана строка $w \in \{0, 1\}^{A(n)}$. (Мы допускаем, что n может быть определено из A(n).)

1. Запустить $A(1^n)$, чтобы получить пару сообщений $m0, m1 \in \{0, 1\}^{A(n)}$.

2. Выбрать единообразный бит $b \in \{0, 1\}$. Задать $c := w \oplus mb$.

3. Передать c A и получить выходные данные b^r. Вывести 1, если $b^r = b$, и 0 в противном случае.

Очевидно, что D работает в полиномиальном времени (предполагается, что A работает в полиномиальном времени).

Прежде чем анализировать поведение D, мы определим измененную систему шифрования $\Pi\tilde{} = (G\tilde{}en, E\tilde{}nc, D\tilde{}ec)$, которая с точностью соответствует системе шифрования Вернама за исключение того, что мы теперь включаем параметр

безопасности, который определяет длину зашифрованного сообщения. Иными словами, G˜en(1n) выводит унифицированный ключ k длиной A(n), и шифрование сообщения $m \in 2^{A(n)}$ с использованием $k \in \{0, 1\}^{A(n)}$ является шифртекстом c := k⊕m. (Расшифрование может осуществляться как обычно, но для нас это сейчас не так важно.) Совершенная стойкость шифра Вернама подразумевает

$$\Pr\left[\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\widetilde{\Pi}}(n) = 1\right] = \frac{1}{2}.$$

Чтобы проанализировать поведение D, проведем следующие основные наблюдения:

1. Если w равномерно выбрана из $\{0, 1\}^{A(n)}$, тогда вид A, когда он работает в качестве подпрограммы D, распределен идентично виду A в эксперименте $\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\widetilde{\Pi}}(n)$. Это происходит из-за того, что когда A запускается в качестве подпрограммы D(w) в этом случае, A задается шифртекст $c = w \oplus m_b$, где $w \in \{0, 1\}^{A(n)}$ является однородной. Поскольку D выводит 1 точно тогда, когда A добивается успеха в своем подслушивающем эксперименте, вследствие этого мы имеем (см. уравнение (3.3))

$$\Pr_{w \leftarrow \{0,1\}^{\ell(n)}}[D(w) = 1] = \Pr\left[\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\widetilde{\Pi}}(n) = 1\right] = \frac{1}{2}. \qquad (3.4)$$

(Индекс первой вероятности явно обозначает, что w выбирается равномерно из $\{0, 1\}^{A(n)}$.)

2. Если w, напротив, генерируется выбором однородного $k \in \{0, 1\}^n$ и последующим заданием w := tt(k), вид A, работающего в качестве подпрограммы D, распределяется идентично виду A в эксперименте $\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\widetilde{\Pi}}(n)$. Это из-за того, что A, работающему в качестве подпрограммы D, теперь задан шифртекст c = w⊕mb, где w = tt(k) для однородного $k \in \{0, 1\}^n$. Таким образом,

$$\Pr_{k \leftarrow \{0,1\}^n}[D(G(k)) = 1] = \Pr\left[\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1\right]. \qquad (3.5)$$

Поскольку tt - псевдослучайный генератор (и D работает в полиномиальном времени), мы знаем, что существует такая пренебрежимо малая функция negl, что

$$\left|\Pr_{w \leftarrow \{0,1\}^{\ell(n)}}[D(w) = 1] - \Pr_{k \leftarrow \{0,1\}^n}[D(G(k)) = 1]\right| \leq \mathsf{negl}(n).$$

Используя уравнения (3.4) и (3.5), мы видим, что

$$\left|\frac{1}{2} - \Pr\left[\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1\right]\right| \leq \mathsf{negl}(n),$$

которая подразумевает $\Pr\left[\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1\right] \leq \frac{1}{2} + \mathsf{negl}(n)$. Поскольку A является произвольным противником, это завершает доказательство того, что Π обладает неразличимым шифрованием при наличии подслушивающей стороны.

Легко запутаться в подробностях доказательства и задаваться вопросом, получилось ли что-нибудь в сравнении с шифром Вернама, не стоит забывать, что

шифр Вернама также шифрует A-битные сообщения с помощью операции XOR по отношению к A-битным строкам! Смысл конструкции, конечно, заключается в том, что A-битные строки tt(k) могут быть *значительно длиннее* секретного ключа *k*. В частности, вышеупомянутая система позволяет зашифровать файл размером 1 Мб с помощью всего лишь 128-битного ключа. Полагаясь на вычислительную стойкость, мы таким образом обошли невозможность теоремы 2.10, в которой утверждается, что любая совершенно криптостойкая система шифрования должна использовать ключ по крайней мере такой же длины, что и сообщение.

Сокращения - предмет обсуждения. Мы безусловно не доказываем, что конструкция 3.17 криптографически надежна. Скорее, мы доказываем, что она криптографически надежна *при допущении*, что tt - псевдослучайный генератор. Такой подход *сокращения* надежности конструкции высокого уровня до примитива низкого уровня имеет ряд преимуществ (обсуждение в разделе 1.4.2). Первое преимущество заключается в том, что обычно легче спроектировать примитив низкого уровня, чем примитив высокого уровня; также обычно проще непосредственно проанализировать алгоритм tt с учетом определения низкого уровня, чем проанализировать более сложную систему Π с определением высокого уровня. Это не значит, что спроектировать псевдослучайный генератор «легко». Это значит, что спроектировать его легче, чем собирать систему шифрования с нуля. (В данном случае система шифрования не выполняет ничего кроме операции исключающего ИЛИ в отношении выходных данных псевдослучайного генератора и сообщения, таким образом, это не совсем так. Однако мы рассмотрим более сложные конструкции, и в тех случаях возможность сокращения задач до простых имеет большое значение.) Другое преимущество заключается в том, что, один раз создав подходящий tt, можно использовать его в качестве компонента других различных систем.

Конкретная криптостойкость. И хотя теорема 3.18 и ее доказательство являются асимптотическими, мы без труда можем адаптировать доказательство к условиям конкретной безопасности системы шифрования в отношении конкретной безопасности tt. Зафиксируем некоторое значение n до конца данного обсуждения, пусть Π теперь обозначает конструкцию 3.17, использующую это значение n. Допустим, tt является (t, ε)-псевдослучайным (для заданного значения n) в том смысле, что для всех дистинктором D, работающих максимум t, мы имеем

$$\Pr[D(r) = 1] - \Pr[D(tt(s)) = 1]. \leq \varepsilon. \qquad (3.6)$$

(Представьте, что $t \approx 280$ and $\varepsilon \approx 2^{-60}$, хотя точные значения несущественны для нашего обсуждения.) Мы заявляем, что Π является $(t - c, \varepsilon)$-надежной для некоторой (малой) постоянной c в том смысле, что для всех A, работающих максимум $t - c$, мы имеем

$$\Pr\left[\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\Pi} = 1\right] \leq \frac{1}{2} + \varepsilon. \qquad (3.7)$$

(Обратите внимание, теперь выше приведены фиксированные числа, а не функции от n, поскольку мы больше не находимся в асимптотических условиях.) Чтобы понять это, допустим, A - это произвольный противник, работающий максимум t − c. Дистинктор D, сконструированный в доказательстве теоремы 3.18, помимо работы A имеет очень небольшие производительные издержки; задание с надлежащим образом обеспечивает работу D максимум t. Наше допущение о том, что конкретная надежность tt подразумевает уравнение (3.6); продолжая также, как и в доказательстве теоремы 3.18, мы получаем уравнение (3.7).

3.4 Более сильные понятия криптостойкости

До настоящего момента мы рассматривали относительно слабые определения криптостойкости, в которых противник только пассивно перехватывал отдельные шифртексты, пересылаемые честными участниками. В данном разделе мы рассмотрим два сильных понятия. Напомним, что определение криптостойкости задает цель безопасности и модель атаки. При определении первого нового понятия криптостойкости мы измени цель безопасности, а для второго усилим модель угроз.

3.4.1 Защита многократного шифрования

Определение 3.8 рассматривается в случае, когда общающиеся стороны передают один шифртекст, который изучается подслушивающей стороной. Было бы удобнее, однако, если бы общающиеся стороны могли отправлять друг другу несколько шифртекстов (все сгенерированы с использованием одного ключа), даже если подслушивающая сторона будет анализировать их все. Для таких приложений нам понадобится система шифрования, безопасная для шифрования нескольких сообщений.

Начнем с подходящего определения криптостойкости в таких условиях. Как и в случае определения 3.8 мы сначала проведем соответствующий эксперимент, определенный для любой системы шифрования П, противника A и параметра безопасности n:

Эксперимент по перехватыванию нескольких сообщений $\mathsf{PrivK}^{\mathsf{mult}}_{\mathcal{A},\Pi}(n)$**:**

3.4.1.1 Противнику A на входе дано 1^n, и он выводит пару равных по длине списков сообщений $M^{\cdot} 0 = (m0,1, \ldots, m0,t)$ и $M_1 = (m1,1, \ldots, m1,t)$, при $|m0,i| = |m1,i|$ для всех i.

3.4.1.2 С помощью алгоритма Gen(1^n) генерируется ключ k и выбирается единообразный бит $b \in \{0, 1\}$. Для всех i вычисляется шифртекст $ci \leftarrow Enck (mb,i)$, и список $C^{\cdot} = (c1, \ldots, ct)$ передается A.

3.4.1.3 A выводит бит b^r.

3.4.1.4 Результатом эксперимента является 1, если $b^r =$, иначе 0.

Определение криптостойкости остается прежним за исключением того, что

теперь оно относится к вышеупомянутому эксперименту.

ОПРЕДЕЛЕНИЕ 3.19 *Система шифрования с закрытым ключом П = (Gen, Enc, Dec) обладает неразличимым многократным шифрованием при наличии подслушивающей стороны, если для всех вероятностных полиномиально-временных противников А существует такая пренебрежимо малая функция negl, что*

$$\Pr\left[\mathsf{PrivK}^{\mathsf{mult}}_{\mathcal{A},\Pi}(n) = 1\right] \leq \frac{1}{2} + \mathsf{negl}(n),$$

где вероятность вычисляется по случайности, используемой А, и случайности, используемой в эксперименте.

Любая система шифрования, обладающая неразличимостью многократного шифрования при наличии подслушивающей стороны, очевидно, также соответствует определению 3.8, поскольку эксперимент $\mathrm{PrivK}^{\mathrm{eav}}$ соответствует особому случаю $\mathrm{PrivK}^{\mathrm{mult}}$, где противник выводит два списка, в каждом из которых содержится только одно сообщение. Фактически, наше новое определение строго сильнее определения 3.8, как это показано далее.

ПРЕДПОЛОЖЕНИЕ 3.20 *Существует система шифрования с закрытым ключом, обладающая неразличимостью при наличии подслушивающей стороны, но различимая при многократном шифровании при наличии подслушивающей стороны.*

ДОКАЗАТЕЛЬСТВО Нам не нужно долго искать пример системы шифрования, соответствующей предположению. Шифр Вернама совершенно надежен, а также обладает неразличимым шифрованием при наличии подслушивающей стороны. Мы покажем, что он не надежен в смысле определения 3.19. (Мы уже обсудили эту атаку в главе 2. Здесь же мы просто проанализируем атаку с учетом определения 3.19.)

Точнее, рассмотрим следующего противника А, атакующего систему (в смысле, определенном экспериментом $\mathrm{PrivK}^{\mathrm{mult}}$): А выводит $M^{\cdot}0 = (0^A, 0^A)$ и $M1 = (0^A, 1^A)$. (Первый содержит два одинаковых открытых текста, второй два разных сообщения.) Пусть $C^{\cdot} = (c1, c2)$ - это список шифртекстов, полученных А. Если c1 = c2, тогда А выводит $b^r = 0$; иначе, А выводит $br = 1$. Теперь мы анализируем вероятность того, что $b^r = b$. Важно то, что шифр Вернама детерминирован, таким образом, шифрование одного и того же сообщения (с помощью одного и того же ключа) приводит к появлению одного и того же шифртекста. Итак, если b = 0, тогда c1 = c2, и А в этом случае выводит 0. С другой стороны, если b = 1, тогда каждый раз зашифровывается другое сообщение; следовательно, c1 $\neq$ c2, и А выводит 1. Делаем вывод, что А правильно выводит $b^r = b$ с вероятностью 1, таким образом, система шифрования не является надежной с учетом определения 3.19.

---

[4].В разделе 3.6.1 мы увидим, что если система шифрования отслеживает состояние, тогда возможно надежно зашифровать несколько сообщений даже в случае детерминированного шифрования.

Необходимость вероятностного шифрования. Вышесказанное может выступать в качестве того, что невозможно достигнуть соответствия определению 3.19 с использованием любой системы шифрования. На самом же деле это верно, если система шифрования детерминирована, тогда многократное шифрование одного и того же сообщения (с использованием одного и того же ключа) всегда будет приводить к одному и тому же результату. Это достаточно важно, чтобы сформулировать в качестве теоремы.

ТЕОРЕМА 3.21 *Если П - это (без сохранения состояния4) система шифрования, в которой Enc является детерминированной функцией от ключа и сообщения, тогда П не может обладать неразличимым многократным шифрованием при наличии подслушивающей стороны.*

Это не должно означать, что определение 3.19 слишком сильное. Действительно,утечка информации о том, что два зашифрованных сообщения являются одним и тем же сообщением, - существенное нарушение требований безопасности. (Рассмотрим, например, сценарий, в котором студент зашифровывает ряд ответов верно/неверно!)

Чтобы создать надежную систему шифрования нескольких сообщений, мы должны спроектировать систему со *случайным* шифрованием, таким что одно и то же сообщение, зашифрованное несколько раз, будет соответствовать разным шифртекстам. Это может казаться невозможным, поскольку расшифрование всегда должно быть способно восстановить сообщение. Тем не менее, мы скоро узнаем, как этого добиться.

3.4.2 Атака на основе подобранного открытого текста и защита от атак на основе подобранного открытого текста (CPA-защита)

Атака на основе подобранного открытого текста передает способность перехватчика осуществлять (частичный) контроль за тем, что шифруется честными участниками. Представим ситуацию, в которой два честных участника обмениваются ключом k, а перехватчик может повлиять на то, что эти стороны зашифруют сообщения m1, m2, . . . (с использованием k) и отправят получившиеся шифртексты по каналу, за которым следит перехватчик. Через какое-то время перехватчик рассматривает шифртекст, соответствующий некоторому неизвестному сообщению m, зашифрованного с использованием того же ключа k. Давайте даже предположим, что перехватчик знает, что m - это один из возможных вариантов m0, m1. Защита от атак на основе подобранного открытого текста означает, что даже в этом случае перехватчик не сможет определить, какое из двух сообщений было зашифровано, с вероятностью, значительно превышающей случайную догадку. (Пока что мы вернемся к случаю, когда подслушивающая сторона получила только одно шифрование неизвестного сообщения. В скором времени мы вернемся к рассмотрению случая с несколькими сообщениями.)

Атака на основе подобранного открытого текста в реальных условиях. Атаки на основе подобранного текста представляют реальную опасность? Для начала отметим, что атаки на основе подобранного открытого текста также охватывают (свою особую разновидность) **атаки на основе открытых текстов,** в которых перехватчик знает, какие сообщения были зашифрованы, даже если он и не отбирал их. Более того, в реальных условиях существует несколько сценариев, по которым противник может оказывать значительное влияние на то, какие сообщения будут зашифрованы. Простой пример: перехватчик набирает сообщение на терминале, который в свою очередь шифрует с помощью секретного ключа и отправляет с удаленного (и неизвестного противнику) сервера все, что он набирает. Здесь перехватчик точно контролирует то, что будет зашифровано, но система шифрования должна оставаться надежной, даже если она используется для шифрования (с тем же ключом) данных другого пользователя.

Что интересно, истории известны случаи, когда атака на основе подобранного открытого текста также успешно применялась для взламывания военных систем шифрования. Например, во время Второй мировой войны британские военные размещали мины в определенных местах, зная, что немцы после обнаружения мин зашифруют их расположение и отправят обратно в главное управление. Эти зашифрованные сообщения использовались криптоаналитиками в Блетчли-парк (Bletchley Park) для взламывания немецких систем шифрования. Другим примером служит история, связанная с битвой за Мидуэй (Battle of Midway). В мае 1942 года криптоаналитики ВМС США перехватили японское зашифрованное сообщение, которое сумели частично расшифровать. Оказалось, что японцы планировали атаку на AF, где AF было частью шифртекста, которую специалисты США не смогли расшифровать. По другим причинам в США были уверены, что целью является остров Мидуэй. К сожалению, все попытки убедить планирующий орган в Вашингтоне в том, что это так, были тщетны: распространенным мнением было, что Мидуэй не может быть целью. Криптоаналитики ВМС разработали следующий план: Они приказали ВС США на острове отправить фальшивое сообщение о том, что у них недостаточный запас пресной воды. Японцы перехватили это сообщение и немедленно доложили своему начальству о «Неполадках с водоснабжением на AF». Криптоаналитики ВМС получили доказательства того, что AF соответствовало Мидуэю, и США направили три авианосца в ту область. В результате Мидуэй был спасен, а японские войска понесли значительные потери. Эта битва стала поворотной точкой в войне между США и Японией на Тихом океане.

Криптоаналитики ВМС провели атаку на основе подобранного открытого текста, поскольку могли повлиять (хоть и опосредованно) на то, чтобы японцы зашифровали слово «Мидуэй». Если бы японская система шифрования была защищена от атаки на основе подобранного открытого текста, такая стратегия

американских аналитиков не сработала бы (и история развивалась бы совершенно по-другому)!

Защита от атак на основе подобранного открытого текста CPA-защита . В формальном определении мы моделируем атаки на основе подобранного открытого текста, предоставляя противник A доступ к *оракулу шифрования*, рассматриваемого как «черный ящик», который шифрует сообщения по выбору A с использованием ключа k, неизвестного A. Иными словами, представим, что A имеет доступ к «оракулу» Enck (•). Когда A делает запрос оракулу, предоставляя ему на входе сообщение m, оракул возвращает шифртекст c ← Enck (m) в качестве ответа. (Когда Enc рандомизирован, оракул использует свежую случайную последовательность каждый раз, когда отвечает на запрос.) Противнику позволено взаимодействовать с оракулом шифрования столько раз, сколько необходимо.

Рассмотрим следующий эксперимент, определенный для любой системы шифрования Π = (Gen, Enc, Dec), противника A и параметра безопасности n:

Эксперимент по неразличимости для атак на основе подобранного открытого текста $\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\Pi}(n)$**:**

1. С помощью алгоритма Gen(1^n) генерируется ключ k.

2. Противнику A на входе дается 1^n и доступ к оракулу Enc_k (•), и он выводит пару сообщений m0, m1 одинаковой длины.

3. Выбирается единообразный бит b ∈ {0, 1}, затем вычисляется шифртекст c ← Enck (mb) и передается A.

4. Противник A сохраняет доступ к оракулу Enck (•) и выводит бит b'.

5. Результатом эксперимента является 1 если $b'=b$, иначе 0. В первом случае мы говорим, что A добился успеха.

ОПРЕДЕЛЕНИЕ 3.22 *Система шифрования с закрытым ключом* Π = (Gen, Enc, Dec) обладает неразличимым шифрованием в условиях атаки на основе подобранного открытого текста, или защищена против атаки на основе подобранного открытого текста, *если для всех вероятностных полиномиально-временных противников A существует пренебрежимо малая функция* negl, *такая что*

$$\Pr\left[\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\Pi}(n)=1\right] \leq \frac{1}{2}+\mathsf{negl}(n),$$

где вероятность вычисляется по случайности, используемой A, а также случайности, используемой в эксперименте.

Защита от атак на основе подобранного открытого текста при многократном шифровании

Определение 3.22 может быть расширено до многократного шифрования таким же образом, как было расширено определение 3.8 для получения определения 3.19, т. е. с использованием списков открытых текстов. Здесь мы используем

другой подход, который несколько проще и обладает преимуществами моделирования перехватчиков, которые могут адаптивно подбирать открытые тексты для шифрования, даже после анализа предыдущих шифртекстов. В данном определении мы дадим перехватчику доступ к «левому или правому» оракулу LRk,b, который, получив на входе сообщения равной длины m0, m1, вычисляет шифртекст с ← Enck (mb) и выдает с. Иными словами, если b = 0, тогда противник получает шифрование «левого» открытого текста, если b = 1, тогда он получает шифрование «правого» открытого текста. Здесь b - случайный бит, выбранный в начале эксперимента; как и в предыдущих определениях, цель перехватчика - угадать b. Оно обобщает предыдущее определение надежности многократных сообщений (определение 3.19), потому что вместо вывода списков (m0,1, ..., m0,t) и (m1,1, ..., m1,t), одно из сообщений которых должно быть зашифровано, перехватчик может последовательно запрашивать LRk,b (m0,1, m1,1), ..., LRk,b (m0,t, m1,t). Оно также включает в себя доступ перехватчика к оракулу шифрования, поскольку перехватчик может просто запросить LRk,b(m, m), чтобы получить Enck (m).

Теперь мы формально определим эксперимент, назвав его экспериментом по LR-оракулу. Пусть П - система шифрования, А - противник, а n - параметр безопасности:

Эксперимент по LR-оракулу $\mathsf{PrivK}^{\mathsf{LR\text{-}cpa}}_{\mathcal{A},\Pi}(n)$**:**

1. С помощью алгоритма Gen(1^n) генерируется ключ k.

2. Выбирается единообразный бит b ∈{0, 1}.

3. На входе противнику А дается 1n и доступ к оракулу LRk,b (•, •), как определено выше.

4. Противник А выводит бит br .

5. Результатом эксперимента является 1 если b^r=b, иначе 0. В первом случае мы говорим, что А добился успеха.

ОПРЕДЕЛЕНИЕ 3.23 *Система шифрования с закрытым ключом* П обладает неразличимым множественным шифрованием в условиях атаки на основе подобранного открытого текста, или является устойчивой к атакам на основе подобранного открытого текста (CPA-устойчивой) при многократном шифровании, *если для всех вероятностных полиномиально-временных противников А существует такая пренебрежимо малая функция* negl, *что*

$$\Pr\left[\mathsf{PrivK}^{\mathsf{LR\text{-}cpa}}_{\mathcal{A},\Pi}(n)=1\right] \le \frac{1}{2}+\mathsf{negl}(n),$$

где вероятность вычисляется по случайности, используемой А, и случайности, используемой в эксперименте.

Как следует из нашего более раннего обсуждения, CPA-защита при многократном шифровании по крайней мере также сильна, как и все предыдущие определения. В частности, если система шифрования с закрытым ключом

устойчива к атакам на основе подобранного открытого текста при многократном шифровании, то она, очевидно, также CPA-устойчива. Важно, что обратное утверждение тоже верно. Иными словами, CPA-защита подразумевает CPA-защита в случае многократного шифрования. (Это контрастирует со случаем подслушивающей стороны. См. предположение 3.20.) Сформулируем следующую теорему без доказательства: похожий результат при использовании открытого ключа доказан в разделе 11.2.2.

ТЕОРЕМА 3.24 *Любая система шифрования с закрытым ключом, устойчивая к атакам на основе подобранного открытого текста, также CPA-устойчива при многократном шифровании.*

Значительное техническое преимущество CPA-защиты: Достаточно доказать, что система CPA-устойчива (при единичном шифровании), и мы без дополнительных усилий докажем устойчивость системы к атакам на основе подобранного открытого текста при многократном шифровании.

Устойчивость к атакам на основе подобранного открытого текста в наши дни является минимальным понятием безопасности, которому должна соответствовать система шифрования. Однако в последнее время распространены более строгие требования к характеристикам безопасности, которые мы обсудим в разделе 4.5.

Сообщения фиксированной длины против сообщения произвольной длины. Другим достоинством работы с определением CPA-защиты является того, что оно позволяет рассматривать сообщения фиксированной длины без потери общности. В частности, имея CPA-устойчивую систему шифрования *фиксированной длины* П = (Gen, Enc, Dec), можно достаточно легко сконструировать CPA-устойчивую систему шифрования $П^r$= (Gen^r, Enc^r, Dec^r) для сообщений *произвольной длины*. Для простоты скажем, П зашифровывает сообщения длиной 1 бит (тем не менее, все, что мы говорим, естественно расширяется без учета длины , поддерживаемой П). Оставим Gen^r таким же, как и Gen. Определим Enc^r для любого сообщения m (обладающего некоторой произвольной длиной A), как Enc^r (m) = Enck (m1), . . . , Enck (mA), где mi означает i-тый бит m. Расшифрование осуществляется естественным пустем. Пr является CPA-устойчивой, если П; доказательство следует из теоремы 3.24.

Существуют более эффективные способы шифрования сообщений произвольной длины, чем настройка системы шифрования фиксированной длины, как указано выше. Мы изучим это позже в разделе 3.6.

3.5 Создание CPA-устойчивых систем шифрования

Перед конструированием систем шифрования, устойчивых к атакам на основе подобранного открытого текста, мы сначала введем важное понятие *псевдослучайных функций.*

3.5.1 Псевдослучайные функции и блочные шифры

Псевдослучайные функции обобщают понятие псевдослучайных генераторов. Теперь вместо того, чтобы рассматривать похожие на случайные строки, мы рассматриваем похожие на случайные функции. Как уже говорилось в обсуждении псевдослучайности, нет смысла говорить о том, что какая-то *фиксированная* функция $f : \{0, 1\}^* \rightarrow \{0, 1\}^*$ является псевдослучайной (так же, как и нет смысла говорить, что какая-то фиксированная функция является случайной). Таким образом, мы вместо этого должны ссылаться на псевдослучайность распределения по функциям. Такое распределение естественно осуществляется при рассмотрении функции с ключом, определенной далее.

Функция с ключом $F: \{0, 1\}^* \times \{0, 1\}^* \rightarrow \{0, 1\}^*$ является функцией двух переменных, где первая переменная называется ключ и обозначается k. Мы говорим, F является эффективной, если существует полиномиально-временной алгоритм, который вычисляет F (k, x) заданных k и x. (Нас будут интересовать только эффективные функции с ключом.) При типичном использовании ключ k выбирается и фиксируется, и тогда нас интересует только функция с одной переменной $Fk: \{0, 1\}^* \rightarrow \{0, 1\}^*$, определенная $Fk (x) = F (k, x)$. Ключ параметр безопасности n определяет длину ключа, входную и выходную длину.

Иначе говоря, мы ставим в соответствие с F три функции Akey, Ain, и Aout; для любого ключа $k \in \{0, 1\}^{Akey(n)}$, функция Fk определяется только для входных данных $x \in \{0, 1\}^{Ain(n)}$, в случае которых $Fk(x) \in \{0, 1\}^{Aout(n)}$. Если не указано иное, для простоты мы допустим, что F является сохраняющей длину, что означает, что $Akey (n) = Ain(n) = Aout(n) = n$. Иными словами, фиксируя ключ $k \in \{0, 1\}^n$, мы получаем функция Fk, преобразующую n-битные входные строка в n-битные выходные.

Функция с ключом F обуславливает естественное распределение по функциям, заданных выбором унифицированного ключа $k \in \{0, 1\}^n$, и рассмотрение получившейся функции с одной переменной Fk . Мы называем F псевдослучайной, если функция Fk (для унифицированного ключа k) неотличима от функции, случайно и равномерно выбранной из множества всех функций, обладающих одинаковыми областями определения и значения. Говоря иначе, если ни один эффективный противник не может различить (в смысле, который мы более точно обозначим ниже), взаимодействует он с Fk (для унифицированного k) или f (где f равномерна выбрана из множества всех функций, преобразующих n-битные входные данные в n-битные выходные данные).

Поскольку случайный выбор функции интуитивно представить гораздо сложнее, чем случайный выбор строки, на этом стоит остановиться подробнее. Рассмотрим множество всех функций, преобразующих n-битные входные данные в n-битные выходные, Funcn. Это множество конечно, и выбор функции, преобразующей n-битные входные данные в n-битные выходные данные, означает

равномерный выбор элемента из этого множества. Насколько велико Funcn? Функция f определяется заданием значения в каждой точке своей области определения. Мы можем рассматривать любую функцию (в конечной области определения), как большую таблицу соответствия, в которой хранятся f(x) в строке таблицы, помеченной x. Для $f \in Func_n$ таблица соответствия для f имеет 2^n строк (по одной для каждой точки области определения $\{0, 1\}^n$), где в каждой строке таблицы находится n-битная строка (поскольку область значений f составляет $\{0, 1\}^n$). Объединив все данные таблицы, мы видим, что любая функция из множества Funcn может быть представлена строкой длиной $2^n \cdot n$. Кроме того, это соответствие является одним-к-одному, поскольку каждая строка длиной $2^n \cdot n$ (т. е. каждая таблица, содержащая в себе 2n записей длиной n) определяет уникальную функцию множества $Func_n$. Таким образом, размер Funcn точно равен n числу строк длиной $n \cdot 2n$, или $|Funcn| = 2^{n \cdot 2}$.

Рассмотрение функции в качестве таблицы соответствий позволяет по-другому представлять выбор равномерной функции $f \in Func_n$: Он с точностью соответствует равномерному выбору каждой строки таблицы соответствия f. В частности, это означает, что значения f(x) и f(y) (для любых двух переменных x f= y) являются равномерными и независимыми. Мы можем рассматривать эту таблицу соответствий как заранее наполненную случайными данными , перед оценкой f по любой переменной, или можем рассматривать данные таблицы, как равномерно и оперативно выбранные по мере необходимости, всякий раз, когда f оценивается по новой переменной, по которой f ранее не оценивалась.

Возвращаясь к нашему обсуждению псевдослучайных функций, напомним, что псевдослучайная функция - это функция с ключом F, такая что Fk (для $k \in \{0, 1\}^n$, случайно и равномерно выбранного) не отличима от функции f (для $f \in Func_n$, случайно и равномерно выбранной). Первая выбирается из распределения в пределах (максимум) 1^n различных функций, тогда как вторая выбирается из всех $2n \cdot 2$ функций в Funcn. Несмотря на это, «поведение» этих функций должно выглядеть одинаково для любого полиномиального дистинктора.

Первой попыткой формализации понятия псевдослучайной функции было бы действовать аналогично Определению 3.14. То есть, мы могли бы потребовать, чтобы каждый полиномиальный дистинктор D, который получает описание псевдослучайной функции Fk, выдает 1 с «почти» той же вероятностью, как когда он получает описание произвольной функции f. Тем не менее, это определение неправильно, так как описание произвольной функции имеет экспоненциальную длину (данную ее таблицей длиной $n \cdot 2n$), в то время как D ограничен работой в полиномиальном времени. Таким образом, D даже не будет располагать достаточным временем, чтобы изучить все входные данные.

Таким образом, определение дает D доступ к оракулу O, который либо равен Fk (для постоянной k), либо f (для постоянной функции f). Дистинктор D может

посылать запросы своему оракулу в любой точке x, в ответ на что оракул возвращает O(x). Мы рассматриваем оракул как черный ящик в той же манере, как когда мы давали противнику оракульный доступ к алгоритму шифрования в определении атаки с выбором незашифрованного текста. Однако, здесь оракул вычисляет детерминированную функцию и, таким образом, выдает тот же результат, если запрос делается дважды с одинаковыми входными данными. (По этой причине, мы можем предположить без ущерба общности утверждения, что D никогда не посылает запрос оракулу дважды с одинаковыми исходными данными.) D может свободно взаимодействовать со своим оракулом, выбирая свои запросы адаптивно на основе всех предыдущих результатов. Однако, так как D работает в полиномиальном времени, он может сделать только полиномиальное количество запросов . Представим теперь формальное определение. (Исключительно с целью упрощения это определение предполагает, что F не меняет длину.)

ОПРЕДЕЛЕНИЕ 3.25 *Пусть F :{0, 1}*×{0, 1}*→{0, 1}* будет эффективной, сохраняющей длину функцией с ключом. F является псевдослучайной функцией если для всех вероятностных полиномиальных дистинкторов D, существует пренебрежимо малая функция negl, такая что*:

$$\left|\Pr[D^{F_k(\cdot)}(1^n)=1]-\Pr[D^{f(\cdot)}(1^n)=1]\right|\le \mathsf{negl}(n),$$

где первая вероятность вычисляется над равномерно выбранными k∈ {0, 1}^{n} и произвольным D, а вторая вероятность вычисляется над равномерно выбранными f ∈Funcn и и произвольным D.

Важно отметить, что дистинктору D не дается ключ k. Требование о псевдослучайности Fk не имеет смысла, если нам исвестна k , так как при известной k задача отличить оракул для Fk от оракула для f является тривиальной. (Все, что требуется от дистинктора, это послать запрос оракулу в любой точке x , чтобы получить ответ y, и сранить полученное с результатом y^r := Fk (x), который он вычисляет сам с помощью известного значения k. Оракул для Fk вернет y = y^r, тогда как оракул для произвольной функции будет иметь y = yr с вероятностью лишь 2−n.) Это означает, что если k известна, никакие утверждения о псевдослучайности Fk не имеют силу. Рассмотрим конкретный пример. Если F — произвольная функция, то при оракульном доступе к функции Fk (для единообразной k) должно быть сложно найти такой входной параметр x , для которого Fk (x) = 0n (так как было бы сложно найти такой входной параметр для действительно произвольной функции f). Но если k известна, найти такой параметр может быть просто.

Пример 3.26

Как полагается, мы можем лучше ознакомиться с определением, рассмотрев нестойкий пример. Определим сохраняющей длину функцию с ключом F как F (k, x) = k⊕x. Для любого входного x, значение Fk (x) равномерно распределено

(когда k единообразна). Несмотря на это, F не является псевдослучайной, так как ее значения в любых двух точках не взаимосвязаны. Конкретнее, рассмотрим дистинктор D, который дает запросы оракулу O в произвольных, различных точках x1, x2 , и получет значения $y1 = O(x1)$ и $y2 = O(x2)$, и выдает 1 тогда и только тогда, когда $y1 \oplus y2 = x1 \oplus x2$. Если $O = Fk$ для любой k, то D выдает 1. С другой стороны, если $O = f$ для f равномерно выбранной из Funcn, то вероятность того, что $f(x1) \oplus f(x2) = x1 \oplus x2$ равна вероятности того, что $f(x2) = x1 \oplus x2 \oplus f(x1)$, или 2^{-n}, а D выдает 1 с этой же самой вероятностью. Разность равна $|1 - 2^{-n}|$, то есть она не пренебредимо мала. ♦

Псевдослучайные перестановки/Блочные шифры

Пусть $Perm_n$ — это множество всех перестановок (т.е. взаимно-однозначных соответствий) над $\{0, 1\}^n$. Как и ранее, мы рассматриваем любую $f \in Perm_n$ как таблицу, но теперь мы добавили ограничение, что входные данные в любых двух различных строках таблицы должны различаться. Скажем, есть 2n различных возможных выборов для данных первой строки таблицы; как только эта строка выбрана, у нас остается только $2n - 1$ возможных выборов для второй строки, и так далее. Таким образом, мы видим, что размер $Perm_n$ равен $(2^n)!$.

Пусть F — функция с ключом. Назовем *F перестановкой с ключом*, если Ain= Aout, и кроме того, для всех $k \in Akey(n)$ функция $Fk: \{0, 1\}^{Ain(n)} \rightarrow \{0, 1\}^{Ain(n)}$ является взаимно однозначной (т.е. Fk является перестановкой). Назовем Ain блочной длиной функции F. Как и ранее, если не указано иначе, предполагается, что является сохраняющей длину и, таким образом, Akey(n)=Ain(n)=n. Перестановка с ключом является эффективной, если существует алгоритм полиномиального времени для вычисления Fk(x) при заданных k и x, а также существует алгоритм полиномиального времени для вычисления $F^{-1}(y)$ при заданных k и y. То есть, Fk должна быть одновременно эффективно вычисляемой и э*ффективно обратимой* при известной k.

Определение того, что значит для еффективной перестановки с ключом F быть *псевдослучайной перестановкой* абсолютно аналогично Определению 3.25, с единственным отличием в том, что теперь Fk должна быть неотличима от равномерной перестановки ,а не от равномерной функции. Таким образом, мы задаем требование, что никакой эффективный алгоритм не может отличить доступ к Fk (для единообразного ключа k) от доступа к f (для однородной $f \in Perm_n$). Оказывается, что это всего лишь решение эстетического характера, так как, когда длина блока достаточно велика, произвольная пер естановка сама по себе неотличима от произвольной функции. Интуитивно, это происходит потому, что равномерная функция f выглядит идентично равномерной перестановке, однако только в том случае, если не существует таких значений x и y, для которых $f(x) = f(y)$, так как в этом случае эта функция не может быть перестановкой. Однако, вероятность найти такие значения x, y с помощью по-

линомиального числа запросов пренебрежимо мала. (Это следует из результатов Приложения А.4.)

Предположение 3.27Если F — псевдослучайная перестановка и, к тому же, Ain(n) ≥ n, то F также является псевднопроизвольной функцией.

Если F — это перестановка с ключом, то криптографические схемы, основанные на F могут предполагать наличие честных участников для вычисления обратной функции F −1 в добавление к вычислению самой функции Fk . Это потенциально предполагает новые проблемы с безопасностью. В частности, в этом случае, возможно, необходимо ввести более сильное требование, а именно, что Fk должна быть неотличимой от равномерной перестановки даже если дистинктору дополнительно предосталяется оракульный доступ к обратной функции этой перестановки. Если F обладает этой характеристикой, то мы назовем ее сильной псевдослучайной перестановкой.

ОПРЕДЕЛЕНИЕ 3.28 Пусть F:{0, 1}×{0, 1}*→{0, 1}* будет эффективной, сохраняющей длину перестановкой с ключом. F является сильной псевдослучайной перестановкой, если для всех вероятностных полиномиальных дистинкторов D, существует пренебрежимо малая функция* negl*, такая что:*

$$\left|\Pr[D^{F_k(\cdot),F_k^{-1}(\cdot)}(1^n)=1]-\Pr[D^{f(\cdot),f^{-1}(\cdot)}(1^n)=1]\right|\le \mathsf{negl}(n),$$

где первая вероятность вычисляется над равномерно выбранными k∈ {0,1}n и произвольным D, а вторая вероятность вычисляется над равномерно выбранными f∈Permn и и произвольным D.

Очевидным образом, любая сильная псевдослучайная перестановка является также псевдослучайной перестановкой.

Блочные шифры. На практике, блочные шифры создаются как защищенные экземпляры (сильных) псевдослучайных перестановок с некоторой фиксированной длиной ключа и длиной брока. Мы обсудим различные подходы к созданию блочных шифров, а также некоторые популярные варианты блочных шифров в Главе 6. Для целей текущей главы детали конструкций неважны, и на данный момент мы просто предположим, что (сильные) псевдослучайные перестановки существуют.

Псевдослучайные функции и псевдопроизвольные генераторы. Как можно ожидать, существует тесная связь между псевдослучайными функциями и прсевдопроизвольными генераторами. Построить псевдослучайный генератор tt из псевдослучайной функции F достаточно легко, просто вычислив F на серии различных аргументов. Например, мы можем определить $G(s) \stackrel{\text{def}}{=} F_s(1)\|F_s(2)\|\cdots\|F_s(\ell)$ для любой A. Если бы вместо Fs была равномерная функция f , результат tt был бы равномерным; таким образом, при использовании вместо этого F , результат получается псевдослучайным. Вам нужно будет

формально доказать это утверждение в Упражнении 3.14 .

В более общем плане, мы можем использовать эту идею для построения поточного шифра (Init, GetBits) который принимает синхропосылку IV . (См. Раздел 3.3.1) Единственное отличие состоит в том, что вместо вычисления Fs на фиксированной последовательности входных данных 1, 2, 3, . . ., мы вычисляем F на данных IV + 1, IV + 2, ...

КОНСТРУКЦИЯ 3.29

Пусть F - псевдослучайная функция. Определим поточный шифр (Init, GetBits) где каждых вызов GetBits выдает n битов, следующим образом:

• Init на входе $s \in \{0,1\}^n$ и $IV \in \{0,1\}^n$ так st0 := (s, IV).

• GetBits: на входе st_i=(s, IV) на выходе IV^r=IV+1и y:= Fs(IV r) и sti+1 := (s, IV r). Вывести (y, sti+1). так st0 := (s, IV).

Поточный шифр из любой псевдослучайной функции/блочного шифра.

Несмотря на то, что поточные шифры могут быть построены из блочных шифров, специально созданные поточные шифры на практике обычно более эффективны, особенно в условиях ограниченных ресурсов. С другой стороны, судя по всему, поточные шифры менее понятны (на практике), чем блочные шифры, и поэтому уверенность в их безопасности меньше. Поэтому, рекомендуется по мере возможности использовать блочные шифры (возможно, сначала преобразовывая их в поточные).

Рассмотрим противоположное направление. псевдослучайный генератор tt незамедлительно выдает псевдослучайную функцию *F с малой длиной блока.* Более конкретно, предположим, что tt имеет коэффициент расширения $n \bullet 2^{t(n)}$. Мы можем дать определение функции с ключом $F : \{0,1\}^n \times \{0,1\}^{t(n)} \rightarrow \{0,1\}^n$ следующим образом: чтобы вычислить Fk (i), сначала вычислить tt(k) и представить результат как таблицу с количеством строк $2^{t(n)}$, каждая из которых содержит n битов; вывести i-ую строку. Вычисление происходит в полиномиальном времени, только если t(n) = O(log n). Также можно, хотя сложнее, построить псевдослучайные функции с *большой блочной длиной* из псевдослучайных генераторов; это показано в Разделе 7.5. псевдослучайные генераторы, в свою очередь, могут быть построены на основе определенных предположительно сложных математических задач. Существование псевдослучайных функций, основанных на таких сложных математических задачах, представляют одно из поразительных достижений современной криптографии.

3.5.2 Шифрование, защищенное от атак с выбором открытого текста, из псевдослучайных функций

Рассмотрим построение шифровальной схемы с фиксированной длиной, защищенной от атак с выбором открытого текста, (ABOT). Принимая во внима-

ние сказанное в Разделе 3.4.2, это подразумевает существование шифровальной схемы, защищенной от АВОТ для сообщений произвольной длины. В Разделе 3.6 мы обсудим более эффективные методы шифрования сообщений произвольной длины.

Наивной попыткой построить защищенную шифровальную схему из псевдослучайной перестановки было бы определить Enck(m) = Fk (m). Хотя мы считаем, что «никакой информации о m не выявляется» (так как, если f является однородной функцией, то f (m) — просто однородная n-битная последовательность), этот метод шифрования детерменированный, и следовательно не может быть защищенным от АВОТ. В частности, при шифровании одного и того же открытого текста дважды, результатом будет один и тот же шифртекст.

Наша защищенная конструкция рандомизируется. Точнее, шифрование происходит путем применения псевдослучайной функции к произвольному значению r (а не к сообщению) и последующим объединением результата и открытого текста операцией исключающего ИЛИ (XOR). (См. Рисунок 3.3 и Конструкцию 3.30.) Опять-таки, это может считаться образцом объединения псевдослучайной ленты с открытым текстом операцией XOR, с главной разницей в том, что каждый раз используется новая псевдослучайная лента. (В действительности, псевднослучайная лента может быть «новой», только если псевдослучайная функция применяется к «новому» знвчению, к которому она никогда прежде не применялась. Хотя и возможно, что произвольное r будет равно какому-либо значению r, которое уже было выбрано ранее, это может случиться только с пренебрежимо малой вероятностью.)

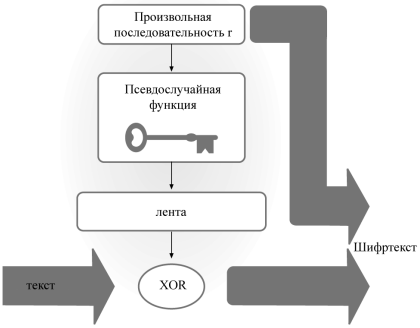

Рисунок 3.3: Шифрование с псевдослучайной функцией

Доказательства безопасности, основанные на псевдослучайных функциях. Прежде чем обратиться к доказательству того, что вышеупомянутая конструкция является защищенной от АВОТ, мы рассмотрим общий шаблон, используемый в большинстве доказательствах защищенности (даже вне контекста шифрования) для конструкций на основе псевдослучайных функций.

В качестве первого шага, в таких доказательствах обычно рассматривается гипотетическая версия конструкции, в которой псевдослучайная функция заменена на случайную. Затем утверждается, используя доказательство от противного, что эта модификация не влияет значительным образом на вероятность успешной атаки. Тогда у нас остается схема, использующая абсолютно случайную функцию. Таким образом, оставшаяся часть доказательства обычно основывается на анализе теории вероятностей и не зависит ни от каких вычислительных утверждений. Мы еще несколько раз используем этот шаблон для доказательсв в этой и следующей главах.

КОНСТРУКЦИЯ 3.30

Пусть F — псевдослучайная функция. Определим шифровальную схему с закрытым ключом для сообщений длины n следующим образом:

• Gen: для входа 1^n, выбери и выведи $k^n \in 0, 1\}$.

• Enc: для входа — ключа $k \in \{0, 1\}^n$ и сообщения $m \in \{0, 1\}^n$, выбери равномерную $r \in \{0,1\}^n$ и выведи шифртекст c := (r, Fk(r)⊕m).

•Dec: для входа — ключа $k \in \{0, 1\}$ и шифртекста c = (r, s), выведи сообщение с открытым текстом m := Fk (r)⊕s.

Шифровальная схема из любой псевдослучайной функции, защищенная от АВОТ.

ТЕОРЕМА 3.31 *Если F — псевдослучайная функция, то Конструкция 3.30 является шифровальной схемой с закрытым ключом, защищенной от АВОТ, для сообщений длины n.*

ДОКАЗАТЕЛЬСТВО Пусть $\tilde{\Pi} = (\tilde{G}en, \tilde{E}nc, \tilde{D}ec)$ — шифровальная схема, абсолютно идентичная схеме Π = (Gen, Enc, Dec) из Конструкции 3.30, за исключением того, что вместо функции Fk используется случайная функция f. То есть, $\tilde{G}en(1n)$ выбирает равномерную функцию $f \in Func_n$, и $\tilde{E}nc$ шифрует точто так же, как Enc, только вместо Fk используется f. (Эта модифицированная шифровальная схема неэффективна. Но мы все же можем определить ее как гипотетическую шифровальную схему для доказательства.)

Зафиксируем произвольного ppt-противника A, и пусть q(n) будет максимальным значением из числа запросов, которые посылает A(1n) своему шифровальному оракулу. (Заметим, что q должен иметь полиномиальное максимальное значение.) В качестве первой ступени доказательства, мы покажем, что существует пренебрежимо малая функция negl , такая что

$$\left|\Pr\left[\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\Pi}(n)=1\right]-\Pr\left[\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\tilde{\Pi}}(n)=1\right]\right|\leq \mathsf{negl}(n). \qquad (3.8)$$

Используем доказательство от противного. Используем A , чтобы построить дистинктор D для псевдослучайной функции F . Дистинктор D получает оракульный доступ к некоторой функции O, и его задача состоит в том, чтобы определить, является ли эта функция «псевдослучайной» (т.е. равной Fk для равномерной $k \in \{0, 1\}^n$) или «случайной» (т.е. равной f для равномерной $f \in Func_n$).

Для этого, D эмулирует эксперимент PrivKcpa для A, как описано ниже, и наблюдает, справится ли A. Если A справляется, то D заключает, что его оракул является псевдослучайной функцией, тогда как если A не справляется, то D заключает, что его оракул является случайной функцией. Более подробно:

Дистинктор D:

D получает вход 1^n и доступ к оракулу O: $\{0, 1\}^n \rightarrow 0, 1\}^n$.

1. Запустить A(1n). Когда A посылает запрос своему шифровальному оракулу о сообщении $m \in \{0, 1\}^n$, ответить на запрос следующим образом:

(a) Выбрать равномерную $r \in \{0, 1\}^n$.

(b) Сделать запрос O(r) и получить ответ y.

(c) Вернуть A шифртекст (r, y⊕m) .

2. Когда A выдает сообщения m0, m1 $\in \{0, 1\}^n$, выбрать единообразный бит $b \in \{0, 1\}$ и затем:

(a) Выбрать равномерную $r \in \{0, 1\}n$.

(b) Сделать запрос O(r) и получить ответ y.

(c) Вернуть A анализируемый шифртекст (r, y⊕mb) .

3. Продолжить таким же образом отвечать на запросы A к шифровальному оракулу, пока A не выведет бит b^r. Вывести 1, если $b^r = b$, и 0 в противном случае.

D работает в полиномиальном времени, так как A работает в полиномиальном времени. Основные моменты следующие:

1. Если оракул дистинктора D является псевдослучайной функцией, то когда A работает как подпрограмма D , обзор A распределен так же, как и при эксперименте $\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\Pi}(n)$. Это происходит потому, что в этом случае ключ k выбран равномерно и произвольно и, таким образом, каждое шифрование происходит путем выбора равномерной r, вычисляя y := Fk (r), и устанавливая шифртексти равный (r, y⊕m), так же как в Конструкции 3.30. Таким образом,

$$\Pr_{k \leftarrow \{0,1\}^n}\left[D^{F_k(\cdot)}(1^n) = 1\right] = \Pr\left[\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\Pi}(n) = 1\right], \qquad (3.9)$$

где мы подчеркиваем, что k выбирается равномерно с левой стороны.

2. Если оракул дистинктора D является случайной функцией, то когда A работает как подпрограмма D , обзор A распределен так же, как и при эксперименте $\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\Pi}(n)$. Это можно рассматривать так же, как приведено выше, с единственной разницей в том, что вместо Fk используется равномерная функция $f \in \mathrm{Func}_n$. Таким образом,

$$\Pr_{f \leftarrow \mathsf{Func}_n}\left[D^{f(\cdot)}(1^n) = 1\right] = \Pr\left[\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\widetilde{\Pi}}(n) = 1\right], \qquad (3.10)$$

где f выбирается равномерно из Funcn с левой стороны.

Следуя из предположения, что F —псевдослучайна функция (и так как D явля-

ется эффективным), существует пренебрежимо малая функция negl , для которой $\left|\Pr\left[D^{F_k(\cdot)}(1^n)=1\right]-\Pr\left[D^{f(\cdot)}(1^n)=1\right]\right|\le \mathsf{negl}(n)$.

Объединение вышеприведенных Уравнений (3.9) и (3.10) дает Уравнение (3.8).

Для следующей части доказательства, продемострируем, что

$$\Pr\left[\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\tilde{\Pi}}(n)=1\right]\le\frac{1}{2}+\frac{q(n)}{2^n}. \qquad (3.11)$$

(Напомним, что q(n) — это предельное значение числа шифровальных запросов, сделанных A. Это утверждение действует, даже если мы не налагаем никаких вычислительных ограничений на A.) Чтобы удостовериться, что Уравнение (3.11) действует, можно пронаблюдать, что, каждый раз, когда сообщение m шифруется в $\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\Pi}(n)$ (либо шифровальным оракулом, либо при вычислении анализируемого шифртекста), выбирается равномерная $r\in\{0,1\}^n$ и шифртекст становится равным $(r, f(r)\oplus m)$. Пусть r* обозначает произвольную последовательность, используемую при генерации анализируемого шифртекста $(r^*, f(r^*)\oplus mb)$. Существуют две возможности:

1. Значение r* никогда не используется при ответе на запросы A к шифровальному оракулу: В этом случае A ничего не узнает о $f(r^*)$ из взаимодействия с шифровальным оракулом (так как f является действительно случайной функцией). Это означает, что A считает, что значение $f(r^*)$, объединенное с mb операцией исключающего ИЛИ, равномерно распределено и независимо от остальной части эксперимента, так что вероятность того, что A выдаст $b'=b$ в этом случае равна 1/2 (как в случае шифра Вермана).

2. Значение r* используется при ответе на хотя бы один из запросов A к шифровальному оракулу: В этом случае, A может легко определить, был ли зашифрован m0 или m1. Это происходит потому, что, если шифровальный оракул когда-либо возвращает шифртекст (r^*, s) в ответ на запрос зашифровать сообщение m, то противник узнает, что $f(r^*)=s\oplus m$.

Однако, так как A делает не больше q(n) запросов к шифровальному оракулу (и, таким образом, максимальное число значений r , используемых при ответе на запросы A к шифровальному оракулу, равно q(n)), и так как r* выбирается равномерно из $\{0,1\}^n$, то максимальная вероятность этого события равна $q(n)/2^n$.

Пусть Repeat обозначает событие, при котором r* используется шифровальным оракулом при ответе на хотя бы один из запросов A. Как только что было объяснено, максимальная вероятность Repeat равна q(n)/2n, а вероятность того, что успешно взломает $\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\Pi}(n)$ в случае, если Repeat не имеет место, равна 1/2. Таким образом:

$$\begin{aligned}
&\Pr[\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\tilde{\Pi}}(n)=1]\\
&\quad=\Pr[\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\tilde{\Pi}}(n)=1\wedge\mathsf{Repeat}]+\Pr[\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\tilde{\Pi}}(n)=1\wedge\overline{\mathsf{Repeat}}]\\
&\quad\le\Pr[\mathsf{Repeat}]+\Pr[\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\tilde{\Pi}}(n)=1\mid\overline{\mathsf{Repeat}}]\ \le\ \frac{q(n)}{2^n}+\frac{1}{2}.
\end{aligned}$$

Объединив вышеприведенное уравнение с Уравнением (3.8), мы видим, что существует пренебрежимо малая функция negl , такая что $\Pr[\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\Pi}(n) = 1] \leq \frac{1}{2} + \frac{q(n)}{2^n} + \mathsf{negl}(n)$. Поскольку q полиномиально, q(n)/2n тоже полиномиально. Кроме того, сумма двух пренебрежимо малых функций тоже пренебрежимо мала, так что существует пренебрежимо малая функция neglr , такая что $\Pr[\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\Pi}(n) = 1] \leq 1 + \text{neglr}(n)$, чем мы завершаем доказательство.

Конкретная криптостойкость. Вышеприведенное доказательство показывает, что

$$\Pr[\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\Pi}(n) = 1] \leq \frac{1}{2} + \frac{q(n)}{2^n} + \mathsf{negl}(n)$$

для некоторой пренебрежимо малой функции negl. Финальное выражение зависит от криптостойкости F как псевдослучайной функции. Это максимальное значение различающей вероятности алгоритма D (который имеет приблизительно то же время работы, что и противник A).

Выражение q(n)/2n представляет максимальное значение вероятности того, что значение, использованное r* для шифрования анализируемого шифртекста было использовано для шифрования какого-либо другого сообщения.

3.6 Режимы использования

Режимы использования предоставляют метод стойкого (и эффективного) шифрования длинных сообщений с помощью поточных или блочных шрифтов.

3.6.1 Режимы использования с поточными шифрами

Конструкция 3.17 предоставляет метод построения шифровальной схемы с использованием псевдослучайного генератора. У этой схемы есть два главных недостатка. Во-первых, как было показано, длина сообщения, которое должно быть зашифровано, должна быть фиксированной и известной заранее. Во-вторых, схема является лишь надежной с точки зрения EAV, и не защищает от CPA (атак с выбором открытого текста).

Поточные шифры, которые можно считать гибкими псевдослучайными генераторами, могут использоваться для исправления этих недостатков. На практике, поточные шифры используются для шифрования в двух разных случаях: *Синхронизованный режим* и *несинхронизованный режим*.

Синхронизованный режим. В этой шифровальной схеме с отслеживанием состояния, отправитель и получатель должны быть синхронизированы в том смысле, что они знают, сколько открытого текста было зашифровано (соответственно, расшифровано) на данный момент. Синхронизованный режим обычно используется в одном сеансе связи между участниками (см. Раздео 4.5.3), где отслеживание состояния приемлемо и сообщения получаются по порядку без каких-либо потерь. Интуиция здесь в том, что генерируется длинный псев-

дослучайный поток, а другая его часть используется для шифровки сообщения. Синхронизация используется для обеспечения правильной расшифровки (т.е. получатель знает, какая часть потока использовалась для шифрования следующего сообщения), и чтобы предотвратить повторное использование порции потока. Теперь мы опишем этот режим более подробно.

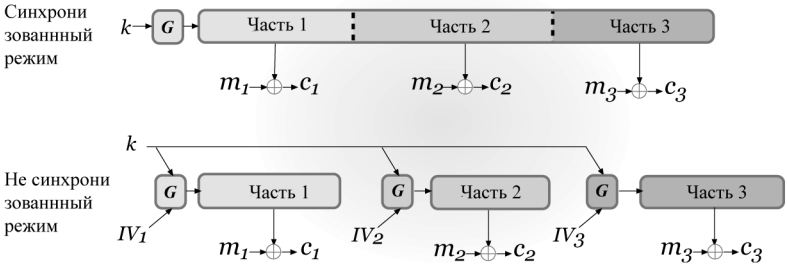

Рисунок 3.4: Синхронизованный и несинхронизованный режим.

Мы видели в Алгоритме 3.16, что поточный шифр может использоваться для построения псевдослучайного генератора ttA с любым желаемым коэффициентом расширения A. Мы можем легко изменить этот алгоритм, чтобы получить псевдослучайный генератор tt∞ с переменной длиной выхода. tt∞ принимает два аргумента: начальное число s и желаемую длину выхода 1A (мы опеределяем это в унарном виде, так как tt будет работать во времени, полиномиальном в A). Как и в Алгоритме 3.16, tt (s, 1A) запускает Init(s) и затем повторно запускает GetBits A раз.

Мы можем использовать tt∞ в Конструкции 3.17 для шифрования сообщений произвольной длины: шифрование сообщения m с помощью ключа k производится вычислением шифртекста c := tt∞(k, 1|)⊕m; расшифровка шифртекста c с помощью ключа k выполняется путем вычисления сообщения m := tt∞(k,1|)⊕c. А изменение доказательства Теоремы 3.18 показывает, что если поточный шифр устойчив, что эта шифровальная схема обладает неразличимым шифрованием при наличии перехватчика.

Немного размышления показывает, что если участники коммуникации хотят сохранить состояние, они мокут использовать один и тот же ключ для шифрования нескольких сообщений. (См. Рисунок 3.4.) Важным замечанием является то, что участники могут рассматривать несколько сообщений m1, m2, . . . как одно длинное сообщение; кроме того, Конструкция 3.17 (также как и измененная версия в предыдущем абзаце) обладает свойством, при котором начальные порции сообщения могут быть зашифрованы и переданы даже если остальная часть сообщения еще не известна. А именно, оба участника имеют ключ k и начинают вычислять st0 := Init(k). Чтобы вычислить первое сообщение m1 длины A1, отправитель повторно запускает GetBits A1 раз, начиная в st0, чтобы полу-

чить поток битов $\mathsf{pad}_1 \stackrel{\text{def}}{=} y_1, \ldots, y_{\ell_1}$ вместе с обновленным состоянием stA1 ; затем он посылает c1:= pad1⊕m1. По получении c1, другой участник повторно запускает GetBits A1 раз, чтобы получить те же значения pad1 и stA1 ; он использует pad1, чтобы восстановить m1:= pad1⊕c1. Затем, чтобы зашифровать второе сообщение m2 длины A2, отправитель повторно запускает GetBits A2 раз, начиная в stA1 , чтобы получить pad2= yA1+1, . . . , yA1+A2 и обновить статус stA1+A2 , а затем вычисляет шифртекст c2:= pad2⊕m2, и так далее. Это может продолжаться неопределенное время, позволяя участникам посылать неограниченное число сообщений произвольной длины. Отметим, что в этом режиме, поточный шифр не должен использовать IV .

Этот метод шифрования нескольких сообщений требует, чтобы участники коммуникации сохраняли синхронизованное состояние, чем объясняется термин «синхронизованный режим». По этой причине, этот метод подходит для тех случаев, когда коммуникация происходит за один «сеанс связи», но не действует правильным образом при единичной коммуникации, или когда один из участников может общаться с разных устройств. (Сохранять копии фиксированного ключа в разных местах довольно легко, однако труднее сохранять сихронизованное состояние в разных местах.) Кроме того, в случае рассинхронизации участников, (например, если одна из передач данных между участниками прервана), расшифровка вернет неправильный результат. Повторная синхронизация возможна, но она прибавляет непроизводительные затраты.

Несинхронизованный режим. Для поточных шифров, у которых функция Init принимает в качестве входа синхропосылку, мы можем получить шифрование, стойкое к атакам с выбором открытого текста (CPA), не сохраняющее информацию о состоянии, для сообщений произвольной длины. Здесь, мы изменим tt∞ , который теперь принимает три аргумента: начальное число s, синхропосылку IV , и желаемую длину выхода 1A. Теперь этот алгоритм сначала вычисляет st0 := Init(s, IV), прежде чем повторно запустить GetBits A раз. Шифрование теперь может быть выполнено, используя вариант Конструкции 3.30: шифрование сообщения m с использованием ключа k выполняется, выбирая равномерную синхропосылку IV∈{0, 1}^{n} и вычисляя m| шифртекст (IV, tt∞(s, IV, 1|)⊕m); расшифровка выполняется обычным образом. (См. Рисунок 3.4.) Эта схема устойчива к атакам с выбором открытого текста (CPA), если поточный шифр теперь обладает более сильным свойством, что для любого полиномиального A функция F, определенная $F_k(IV) \stackrel{\text{def}}{=} G_\infty(k, IV, 1^\ell)$ как Fk (IV)=tt∞(k, IV, 1) является псевдослучайной функцией. (В действительности, F должны быть псевдослучайными, только когда они вычисляются на единообразных аргументах. Функции с ключом с этим более слабым свойством называются слабыми псевдослучайными функциями.)

3.6.2 Режимы использования с блочными шифрами

Мы уже видели конструкцию со схемой, устойчивой к атакам с выбором открытого текста, основанной на псевдослучайной функции/блочных шифрах. Но Конструкция 3.39 (и ее расширенная версия с сообщениями произвольной длины, которая обсуждалась в конце Раздела 3.4.2) имеют недостаток, который состоит в том, что длина шифртекста вдвое больше длины открытого текста. Режимы использования с блочными шифрами предоставляют метод шифрования сообщений произвольной длины, используя более короткие шифртексты.

В этом разделе, пусть F будет блочным шифром длины n. Предположим, что все шифруемые сообщения m имеют длину, кратную n, и будем писать m = m1, m2, . . . , mA , где каждое mi∈ {0, 1}n представляет блок открытого текста. Сообщения, которые не имеют длину кратную n могут быть всегда однозначно дополнены до длины кратной n , добавляя единицу и затем достаточное количество нулей, так что, это утверждение не приносит значительную потерю обобщения.

Известно несколько режимов использования с блочными шифрами. Мы представим четыре из наиболее известных и обсудим их устойчивость.

Режим Книги Электронных Кодов (КЭК). Это родной режим использования, в котором шифртекст получается путем прямого применения блочного шифра к каждому блоку открытого текста. То есть, c := (Fk (m1), Fk (m2), . . . , Fk (mA)); см.

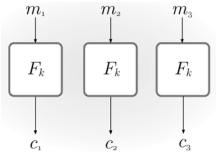

РИСУНОК 3.5: Режим Книги Электронных Кодов (КЭК).

Рисунок 3.5: Расшифровка производится тривиальным образом, используя тот факт, что F^{-1} эффективно вычисляема.

Режим КЭК является детерминистским , и следовательно не устойчив к атакам с выбором открытого текста. Кроме того, шифрование в режиме КЭК даже не имеет неотличимые шифрования при наличии перехватчика. Это объясняется тем, что если блок повторяется в открытом тексте, он будет также повторяться и в шифртексте. Таким образом, можно легко отличить шифрование открытого текста, который состоит из двух идентичных блоков, от шифрования открытого текста, состоящего из двух разных блогов. Это проблема не только теоретического характера. Рассмотрим шифрование изображения, в котором маленькие группы пикселей соответсвуют блоку открытого текста. Шифрование с использованием режима КЭК может раскрыть значительное количество информации о структуре изображения, что не должно происходить при использовании устойчивой схемы

шифрования. Это продемонстрировано на Рисунке 3.6.

По этой причине, режим КЭК никогда не должен использоваться. (Мы рассмотрели его только из-за его исторического значения.)

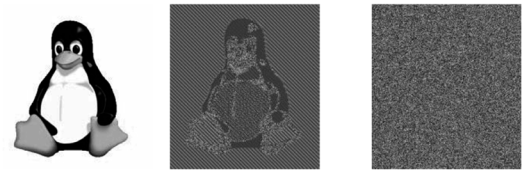

РИСУНОК 3.6 : Иллюстрация опасностей в использовании режима КЭК. Фигура посередине представляет шифрование изображения слева с использованием режима КЭК. Фигура справа представляет шифрование того же изображения с использованием устойчивого режима. (Взято из http://en.wikipedia.org и преобразовано из изображений, созданных Ларри Эрвигом (lewing@isc.tamu.edu) с помощью GIMP.)

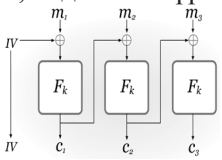

РИСУНОК 3.7: Режим сцепления шифрованных блоков (СШБ).

Режим сцепления шифрованных блоков (СШБ). Для шифрования с использованием этого режима сначала выбирается равномерная синхропосылка (IV) длины n. Затем, блоки шифртекста генерируются путем применения блочного шифра к результату объединения данного блока открытого текста и предыдущего блока шифртекста путем операции исключающего ИЛИ. То есть, установить c0:=IV и затем для i=1 до A, установить $c_i := F_k(c_{i-1} \oplus m_i)$. Финальный шифртекст равен (c0, c1, . . . , cA). (См. Рисунок 3.7.) Расшифровка шифртекста c0, . . . , cA производится путем вычисления $m_i := F_k^{-1}(ci) \oplus c_{i\ 1}$ для i = 1, . . . , A. Подчеркнем, что IV включена в шифртекст. Это очень важно для того, чтобы расшифровка выполнялась.

Также важно, что шифрование в режиме СШБ вероятностное и доказано, что если F — псевдослучайная перестановка, то шифрование в режиме СШБ устойчиво к атакам с выбором открытого текста. Главным недостатком этого режима является то, что шифрование должно производиться последовательно, потому что необходим блок ci−1 шифртекста, чтобы зашифровать блок mi открытого текста. Таким образом, если доступна параллельная обработка, режим СШБ может оказаться не самым эффективным выбором.

Может показаться, что достаточно использовать другую IV (вместо произвольной IV) для каждого шифрования, например, сначала использовать IV

= 1, а затем увеличивать IV на один каждый раз когда сообщение шифруется. В Упражнении 3.20 вам нужно будет продемонстрировать, что эта версия шифрования в режиме СШБ не устойчива. Можно также рассмотреть версию шифрования в режиме СШБ с сохранением состояния, которая называется сцепленным режимом СБШ, и в которой последний блок предыдущего шифртекста используется в качестве IV при шифровании следующего сообщения. Это уменьшает полосу частот, поэтому нет необходимости посылать IV каждый раз. См. Рисунок 3.8, где начальное сообщение m1, m2, m3 зашифровано с помощью произвольного IV , а затем второе сообщение m4, m5 зашифровано с помощью c3 в к ачестве IV . (В противном случае, шифрование с помощью режима СШБ без сохранения состояния создало бы новую IV при шифровке второго сообщения.) Сцепленный режим СШБ используется в SSL 3.0 и TLS 1.0.

Может показаться, что сцепленный режим СШБ так же устойчив, как режим СШБ, так как шифрование m1, m2, m3 в сцепленном режиме СШБ и последующее шифрование m4, m5 дает тот же самый шифртекст, что и шифрование единого сообщения m1, m2, m3, m4, m5 в режиме СШБ. Несмотря на это, сцепленный режим СШБ не предоставляет защиту от атак с выбранным открытым текстом. Основанием атаки является то, что противник заранее знает m1 mz m3 m¢ m5

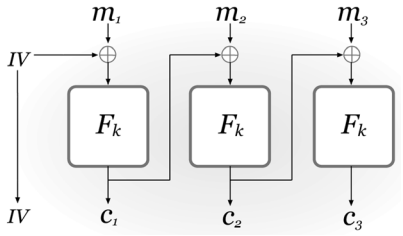

РИСУНОК 3.8 : Сцепленное СШБ.

«синхропосылку», которую мы используем для шифрования второго сообщения. Опишем эту атаку неформально на основе Рисунка 3.8. Предположим, противник знает, что $m1 \in \{m_1^0, m_1^1\}$, и видит первый шифртекст IV, c_1, c_2, c_3. Тогда противник запрашивает шифрование второго сообщения m_4, m_5 с $m_4 = IV \oplus m_1^0 \oplus c_3$, и видит второй шифртекст c_4, c_5. Можно проверить, что $m_1 = m_0$ тогда и только тогда, когда $c_4 = c_1$. Этот пример должен служить сильным предупреждением против внесения любых изменений в шифровальные схемы, даже если они кажутся безвредными.

Режим шифрования с обратной связью по выходу (ОСВ). Третий представляемый режим можно рассматривать как несинхронизованный режим поточных шифров, где поточный шифр строится определенным образом из лежащего в его основе блочного шифра. Мы прямо определим этот режим. Сначала, выбирается равномерная $IV \in \{0, 1\}^n$. Затем псевдослучайный поток генерируется из

IV следующим образом: Определим y0 := IV , и установим i-ый блок yi потока так: yi := Fk (yi−1). Каждый блок открытого текста зашифрован путем его объединения с помощью операции исключающего ИЛИ соответствующим блоком потока. То есть ci := $y_i \oplus m_i$. (См. Рисунок 3.9.) Как и в режиме СШБ, IV является частью шифртекста для обеспечения расшифровки. Однако, в отличие от режима СШБ, здесь не требуется, чтобы F была обратимой. (В действительности, она не должна

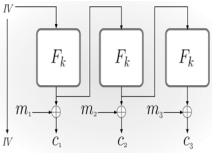

РИСУНОК 3.9: Режим шифрования с обратной связью по выходу (ОСВ).

даже быть перестановкой.) Кроме того, как и в режимах использования с поточными шифрами, здесь не обязательно, чтобы длина открытоко текста была кратной длины блока. Вместо этого, сгенерированный поток может быть сокращен до длины открытого текста. Еще одним преимуществом режима ОСВ является то, что его версия с сохранением состояния (к которой финальное значение yA, используемое для шифрования некоторого сообщения, используется в качестве IV для шифрования следующего сообщения) является устойчивой. Эта версия с сохранением состояния эквивалентна синхронизованному режиму с поточными шифрами, где поточные шифры строятся из блочных шифров только что объясненным способом.

Можно показать, что режим ОСВ является устойчивым, если F — псевдослучайная функция. Хотя шифрование и расшифровка должны производиться последовательно, этот режим имеет преимущество перед режимом СШВ, которое состоит в том, что основной объем вычислений (а именно, вычисление псевдослучайного потока) может производиться независимо от самого сообщение, подлежащего шифрованию. Таким образом, можно сгенерировать псевдослучайный поток заблаговременно, используя предварительную обработку, после чего шифрование открытого текста (как только он известен) происходит невероятно быстро.

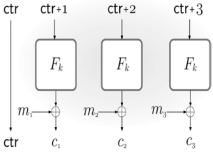

РИСУНОК 3.10: Режим счетчика (CTR).

Режим счетчика (CTR). Режим счетчика можно рассматривать как несинхронизованный режим с поточными шифрами, где поточный шифр строится

из блочного шифра, как было показано в Конструкции 3.29. Приведем ниже замкнутое описание. Для шифрования в режиме счетчика, сначала выбирается равномерное значение $ctr \in \{0, 1\}^n$.

Затем, генерируется псевдослучайный поток путем вычисления $yi := Fk(ctr + i)$, где ctr and i считаются числами сложение выполняется по модулю 2^n. i-ый блок шифртекста равен $ci := yi \oplus mi$, а IV снова посылается как часть шифртекста. См. Рисунок 3.10. Заметим, что расшифровка снова не требует, чтобы F была обратимой, или даже перестановкой. Так же как и в режиме OCB, еще одном режиме «с поточными шифрами», сгенерированный поток может быть сокращен до длины открытого текста, может быть применена предварительная обработка для генерации псевдослучайного потока до того, как известно сообщение, а версия режима счетчика с сохранением состояния является устойчивой.

В отличие от всех устойчивых режимах, обсуждаемых выше, режим счетчика имеет преимущество, состоящее в том, что шифрование и расшифровка могут происходить полностью параллельно, так как все блоки псевдослучайного потока могут быть вычислены независимо друг от друга. В отличие от OCB, здесь также можно расшифровать i-ый блок шифртекста, используя только одно вычисление F. За счет этих характеристик режим счетчика является приемлемым выбором.

Режим счетчика также довольно легко анализировать:

ТЕОРЕМА 3.32 *Если F псевдослучайная функция, то режим счетчика устойчив против атак с выбором открытого текста.*

ДОКАЗАТЕЛЬСТВО Используем тот же шаблон, что и для доказательства Теоремы 3.31: сначала заменим F случайной функцией и затем проанализируем полученную схему. Пусть $\Pi = (Gen, Enc, Dec)$ — шифровальная схема (без сохранения состояния) в режиме счетчика, и пусть $\tilde{\Pi} = (\tilde{G}en, \tilde{E}nc, \tilde{D}ec)$ — шифровальная схема, идентичная Π, с единственным отличием в том, что вместо Fk используется действительно случайная функция. То есть, $\tilde{G}en(1n)$ выбирает равномерную функциюf $\in Func_n$, и $\tilde{E}nc$ шифрует точто так же, как Enc только вместо Fk используется f. (Снова, ни $\tilde{G}en$, ни $\tilde{E}nc$ не являются эффективными, но это не важно для определения эксперимента с $\tilde{\Pi}$.) Зафиксируем произвольного противника ppt A, и пусть q(n) — полиномиальное максимальное значение числа запросов к шифровальному оракулу, сделанных A(1n), а также максимальное число блоков в любом из этих запросов и максимальное число блоков в m0 и m1. В качестве первой ступени доказательства, мы утверждаем, что существует пренебрежимо малая функция negl, такая что

$$\left|\Pr\left[\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\Pi}(n) = 1\right] - \Pr\left[\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\tilde{\Pi}}(n) = 1\right]\right| \leq \mathsf{negl}(n). \qquad (3.12)$$

Это утверждение доказывается от обратного похожим образом, как и аналогичная ступень доказательства Теоремы 3.31, и предосталяется в качестве упражнения читателю.

Затем мы утверждаем, что

$$\Pr\left[\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\tilde{\Pi}}(n)=1\right] < \frac{1}{2}+\frac{2q(n)^2}{2^n}. \qquad (3.13)$$

Соединив это уравнение с Уравнением (3.12), получаем

$$\Pr\left[\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\Pi}(n)=1\right] < \frac{1}{2}+\frac{2q(n)^2}{2^n}+\mathsf{negl}(n).$$

Так как q полиномиально, $\frac{2q(n)^2}{2^n}$ пренебрежительно мало, что и следовало доказать.

Теперь докажем Уравнение (3.13). Зафиксируем некоторое значение n для параметра безопасности. Пусть Let $\ell^* \leq q(n)$ обозначает длину (в блоках) сообщений m0, m1 выданных в эксперименте $\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\Pi}(n)$, и пусть ctr* обозначает начальное значение, использованное при генерации анализируемого шифртекста. Аналогично, пусть Ai ≤ q(n) — длина (в блоках) i-го запроса к шифровальному оракулу, сделанного A, и пусть ctri обозначает начальное значение, использованное при ответе на запрос. Когда дается ответ на i-ый запрос у шифровальному оракулу f применяется в значениям ctri + 1, . . . , ctri + Ai. Когда происходит шифрование анализируемого шифртекста, f применяется к ctr* +1, . . . , ctr* +A*, и i-ый блок шифртекста вычисляется путем объединения операцией исключающего ИЛИ f (ctr* + i) с i-ым блоком сообщения. Есть два случая:

1. Не существуют такие i, j, j*≥1 (с j≤Ai j*≤A*), для которых ctri+j= ctr*+ j*: В этом случае, значения f (ctr* + 1), . . . , f (ctr* + A*), используемые при шифровании анализируемого шифртекста, равномерно распределены и независимы от остальной части эксперимента, так как f не применялась ни к одному из этих аргументов при шифровании запросов противника к оракулу. Это означает, что анализируемый шифртекст вычисляется путем применения операции исключающего ИЛИ к потоку равномерных битов и сообщению mb, то есть вероятность того, что A выдаст br = b равна 1/2 (как в случае шифра Вермана).

2. Существуют i, j, j*≥1 (с j≤Ai иj*≤A*), для которых ctri+j = ctr*+ j* : Мы назовем это событие Overlap. В этом случае, A может потенциально определить, какое из его сообщений было зашифрованно для получения анализируемого шифртекста, так как A может вычислить значение f(ctri +j) = f(ctr*+j*) на основе ответа на его i-ый запрос к оракулу.

Проанализируем вероятность события Overlap . Вероятность максимальна, если A* и каждый Ai максимально велики, так что предположим, что A*= Ai= q(n) для всех i. Пусть Overlapi обозначает событие, при котором последовательность ctri+1, . . . , ctri+q(n) перекрывает последовательность ctr*+1, . . . , ctr*+q(n). Тогда Overlap — это событие, при котором Overlapi имеет место для некоторого i. Так как существует максимум q(n) заросов к оракулу, граница объединения (сравните Предположение A.7) дает

$$\Pr[\mathsf{Overlap}] \leq \sum_{i=1}^{q(n)} \Pr[\mathsf{Overlap}_i]. \qquad (3.14)$$

При фиксировании ctr*, событие Overlapi имеет место именно когда ctri удослетворяет ctr*−q(n)+1≤ctri≤ ct*+q(n)−1.

Так как существует 2q(n) − 1 значений ctri, для которых имеет место Overlapi, и ctri выбирается равномерно из {0, 1}n, мы видим, что

$$\Pr[\mathsf{Overlap}_i] = \frac{2q(n) - 1}{2^n} < \frac{2q(n)}{2^n}.$$

Объединив это уравнение с Уравнением (3.14), получаем Pr[Overlap] < 2q(n)2/2n. Учитывая приведенное выше, мы можем легко оценить вероятность успеха A:

$$\begin{aligned}
\Pr[\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\widetilde{\Pi}}(n) = 1] &= \Pr[\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\widetilde{\Pi}}(n) = 1 \wedge \overline{\mathsf{Overlap}}] \\
&\quad + \Pr[\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\widetilde{\Pi}}(n) = 1 \wedge \mathsf{Overlap}] \\
&\leq \Pr[\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\widetilde{\Pi}}(n) = 1 \mid \overline{\mathsf{Overlap}}] + \Pr[\mathsf{Overlap}] \\
&< \frac{1}{2} + \frac{2q(n)^2}{2^n}.
\end{aligned}$$

Это доказывает Уравнение (3.13) и завершает данное доказательство.

Режимы использования и искажение сообщений. Во многих работах, режимы использования также сравниваются на основе того, как хорошо они защищают от враждебных изменений шифртекста. Мы здесь не приводим такие сравнения, потому что проблема целостности сообщения или подлинности сообщения должна рассматриваться отдельно от шифрования, что мы и сделаем в следующей главе. Ни один из вышеупомянутых режимов не обеспечивает целостность сообщений в значении, которой мы там определим. Отложим последующее обсуждение до следующей главы.

Мы рассмотрим поведение различных режимов в присутствии «безвредных» (т.е. не враждебных) ошибок в передачи данных, см. Упражнение 3.21 и 3.22. Отметим, однако, что в общем такие ошибка могут быть исправлены, используя стандартные методы (например, исправление ошибок или повторная передача).

Длина блока и конкретная стойкость. Режимы СШВ, ОСВ и режим счетчика используют произвольный IV . Это придает эффект рандомизации процесса шифрования и обеспечивает (с высокой вероятностью), что лежащий в основе блочный шифр всегда вычисляется на свежих (т.е. новых) аргументах. Это важно, потому что, как мы видели в доказательствах Теоремы 3.31 и Теоремы 3.32, если аргумент блочного шифра используется более одного раза, безопасность может быть нарушена.

Длина блока блочного шифра, таким образом, имеет значительное влияние на конкретную стойкость схем шифрования на основе этого шифра. Рассмотрим,

например, режим счетчика, использующий блочный шифр F с длиной блока A. Тогда IV — равномерная A-битная последовательность, и мы считаем, что IV повторится после шифрования примерно 2A/2 сообщения (см. Приложение A.4). Если A слишком мала, то, даже если F устойчива как псевдослучайная перестановка, полученная граница конкретной стойкости будет слишком слабой для применения на практике. А имеено, если A = 64, как в случае шифра DES (блочный шифр, который мы рассмотрим в Главе 6), то после 232 ≈ 4,300,000,000 шифрований, или примерно 34 гигабайт открытого текста, ожидается, что IV повторится . Хотя может показаться, что это очень много данных, это меньше, чем объем современных жестких дисков.

Неправильное использование IV. В нашем описании и обсуждении различных (стойких) режимов, мы предполагали, что каждый раз при шифровании сообщения выбирается произвольный IV . Что если это предположение окажется неверным, например, из-за плохой произвольной генерации или ошибочной реализации? Естественно, тогда мы не можем гарантировать устойчивость в значении Определения 3.22. С практической точки зрения, однако, режимы «с поточными шифрами» (ОСВ и режим счетчика) намного хуже, чем СШВ. Если IV повторяется при использовании первых двух режимов, то противник может объединить два полученных шифртекста операцией исключающего ИЛИ и получить много информации о всем содержании обоих зашифрованных сообщений (как мы ранее наблюдали в контексте шифра Вермана с повторным использованием ключа). Однако, в режиме СШВ, существует большая вероятность того, что уже после нескольких блоков ввод блочного шифра «отклонится» и противник не сможет получить ничего больше, чем информацию о нескольких блоках сообщения. Одним из способов избежать неправильное использование IV является использование шифрования с сохранением состояния, как обсуждалось в контексте режимов ОСВ и счетчика. Если шифрование с сохранением состояния невозможно, и существуют подозрения о неправильном использовании IV , то рекомендуется использовать режим СШВ, ввиду причин, описанных выше.

3.7 Атаки с выбором шифртекста

3.7.1 Определение устойчивости против АВШ

До сих пор мы определили устойчивость против двух типов атак: пассивный перехват информации и атаки с выбором открытого текста. *Атаки с выбором шифртекста более действенны.* В атаке с выбором шифртекста, противник имеет возможность не только получить шифрования сообщений на свой выбор (как в атаке с выбором открытого текста), но и расшифровки шифртекстов на свой выбор (с одним исключением, которое мы обсудим ниже). Формально, мы даем противнику доступ к оракулу дешифрования, помимо оракула шифрования. Представим формальное определение о отложим последующее обсуждение.

Рассмотрим следующий эксперимент, определенный для любой системы шифрования с закрытым ключом Π = (Gen, Enc, Dec), противника A, и параметра безопасности n.

Эксперимент неотличимости АВШ $\mathsf{PrivK}^{\mathsf{cca}}_{\mathcal{A},\Pi}(n)$

3.7.1.1 С помощью алгоритма Gen(1n) генерируется ключ k.

3.7.1.2 Противник A получает вход 1n и оракульный доступ к Enck (•) и Deck(•). Он выдает пару сообщений одинаковой длины m0, m1.

3.7.1.3 Выбирается единообразный бит b ∈ {0, 1} , и затем вычисляется шифртекст c ← Enck (mb) , который дается A. Назовем c анализируемым шифртекстом.

3.7.1.4 Противник A продолжает иметь оракульный доступ к Enck (•) и Deck (•), но не может посылать запрос к последнему в отношении самого анализируемого шифртекста. В итоге, A выводит бит b′.

3.7.1.5 Результатом эксперимента является 1, если b′ = b, и 0 в противном случае. Если результатом является 1, мы будем говорить, что A добился успеха.

ОПРЕДЕЛЕНИЕ 3.33 *Шифровальная схема с закрытым ключом* Π имеет неразличимые шифрования при атаке с выбором шифртекста, или *является устойчивой к АВШ, если для всех вероятностных противников, работающих в полиномиальном времени A существует пренебрежимо малая функция* negl, *такая что:*

$$\Pr[\mathsf{PrivK}^{\mathsf{cca}}_{\mathcal{A},\Pi}(n) = 1] \leq \frac{1}{2} + \mathsf{negl}(n),$$

где вероятность вычисляется над всей произвольностью в эксперименте.

В завершение отметим, что естественный аналог Теоремы 3.24 также действует и для устойчивости к АВШ. (А именно, если схема имеет неотличимые шифрования при атаке с выбором шифртекста, то она имеет несколько неотличимых шифрований при атаке с выбором шифртекста, определенных соответствующим образом.)

В рассмотренном выше эксперименте доступ противника к оракулу дешифрования неограничен за исключением ограничения, что противник не может запросить дешифровку самого анализируемого шифртекста. Это ограничение необходимо, так как очевидно, что в противном случае ни одна шифровальная схема не будет соответствовать определению.

В этот момент вы, возможно, сомневаетесь, существуют ли реальные атаки, моделируемые атаками с выбором шифртекста. Как и в случае атак с выбором открытого текста, мы не ожидаем, что честные участники расшифруют произвольные шифртексты по выбору противника. Однако, возможны случаи, когда противник может повлиять на то, что будет расшифровано, и узнать часть информации о результате . Например:

1. В примере про Мидуэй из Раздела 3.4.2 допустимо, что американские криптографы могли попытаться послать зашифрованные сообщения японцам и отследить

их поведение. Такое поведение (например, перемещение военных сил и так далее) могло бы предоставить важную информацию об исходном открытом тексте.

2. Представьте пользователя, посылающего зашифрованное сообщение своему банку. Противник может посылать шифртексты от имени пользователя. Банк расшифрует эти шифртексты, и противник может узнать что-либо о результате. Например, если шифртекст соответствует неправильно созданномуоткрытому тексту (например, непонятное сообщение, или просто неправильное форматирование), противник может понять это по реакции банка (т.е. по модели последующей коммуникации). Практический пример такой атаки представлен ниже в Разделе 3.7.2.

3. Шифрование часто используется в протоколах высшего уровня. Например, шифровальная схема может использоваться как часть протокола аутентификации, где один участник посылает другому шифртекст, и тот его расшифровывает от возвращает результат. В этом случае, один из честных участников действует точно как оракул дешифровки.

Неустойчивость схем, которые мы изучили. Ни одна из шифровальных схем, изученных нами до сих пор, не является устойчивой против АВШ. Продемонстрируем это для Конструкции 3.30, где шифрование вычисляется как $Enc_k(m) = (r, F_k(r) \oplus m)$.

Рассмотрим противника A в эксперименте по неразличимости АВШ, который выбирает $m0 = 0^n$ and $m1 = 1^n$. Тогда, по получении шифртекста $c = (r, s)$, противник может переключить первый бит s и запросить расшифровку остального шифртекста cr. Так как $cr \neq c$, этот запрос разрешен, и оракул дешифровки отвечает либо 10^{n-1} (в этом случае ясно, что $b = 0$) или 01^{n-1} (тогда $b = 1$). Этот пример демонстрирует, что устойчивость против АВШ достаточно строга. Любая шифровальная схема, позволяющая «манипуляцию» шифртекстами в любом контролируемом виде не может быть устойчива против АВШ. Таким образом, устойчивость против АВШ предполагает очень важное свойство, которое называется негибкость. Проще говоря, негибкая шифровальная схема имеет свойство, что если любой противник попытается изменить данный шифртекст, то результатом будет либо недействительный шифртекст, либо шифртекст, расшифровывающий какой-либо открытый текст, не имеющий отношения к исходному. Это очень полезное свойство схем, используемых в сложных криптографических протоколах.

Построение шифровальной схемы, устойчивой против АВШ. Мы покажем, как построить шифровальную схему, устойчивую против АВШ, в Разделе 4.5.4. Построение представлено там, потому что оно использует иструменты, разработанные в Главе 4.

3.7.2 Атаки оракула дополнения

Атака с выбором шифртекста на Конструкцию 3.30, описанная в предыдущем разделе, немного надуманна, так как она предполагает, что атакующая

сторона может получить полное описание измененного шифртекста. Хотя атаки такого типа разрешены Определением 3.33, не очень понятно, являются ли они реальной проблемой на практике. Здесь мы покажем атаку с выбором шифртекста на естественную и используемую повсеместно шифровальную схему. Более того, эта атака лишь требует от атакующей стороны способности определять, правильно или нет расшифрован измененный шифртекст. Эту информацию часто легко получить, так как, например, сервер может запросить повторную передачу или прервать сеанс, если он получает шифртекст, дешифруемый неправильно, и оба этих события произведут заметные изменения в наблюдаемом потоке информации. Было продемонстрировано, что эта атака работает на практике на различных используемых протоколах. Мы приведем один конкретный пример в конце этого раздела.

Как ранее упоминалось, при использовании режима СШВ, длина открытого текста должна быть кратной длине блока. Если открытый текст не удовлетворяет это условие, он должен быть дополнен перед шифрованием. Мы будем называть исходный открытый текст сообщением, а полученный результат после дополнения закодированными данными. Используемая схема дополнения должна позволять получателю однозначно определить, где находится конец закодированных данных. Одним из популярных и стандартных методов является использование дополнения PKCS #5. Предположим, что исходное сообщение содержит целой число байтов, и пусть L обозначает длину блока (в байтах) используемого блочного шифра. Пусть b обозначает число байтов, который должны быть добавлены к сообщению, чтобы общая длина закодированных данных стала кратной длине блока. Здесь, b больше или равно 1 и меньше или равно L. (b не может быть равно 0, так как это приведет к неоднозначному дополнению. Так, если длина сообщения кратна длине блока, добавляются L байтов дополнения.) Затем, мы добавляем к сообщению последовательность, содержащую число b (представленное в 1 байте или 2 шестнадцатеричных числах), повторенное b раз. То есть, если необходим 1 байт дополнения, то добавляется последовательность 0x01 (в шестнадцатеричной системе); если необходимо 4 байта дополнения, то добавляется шестнадцатеричная последовательность 0x04040404, и так далее. Закодированные данные затем шифруются с помощью обычного шифрования в режиме СШВ.

При расшифровке, получатель сначала применяет расшифровку в режиме СШВ для получения закодированных данных, а затем проверяет, правильно ли они дополнены. (Это делается легко: нужно просто прочитать значение b последнего байта, а затем удостовериться, что последние b байтов полученного результата все имеют значение b.) Если это так, то дополнение удаляется и возвращается исходное сообщение. В противном случае, стандартной процедурой является возвращение ошибки «плохое дополнение» (например, в Java стандартное исключение называется **javax.crypto.BadPaddingException**). Наличие такого сообще-

ния об ошибке дает противнике часть оракула дешифровки. То есть, противник может послать серверу любой шифртекст и узнать (на основе того, получил ли он ошибку о «плохом дополнении» или нет), правильным ли образом дополнены исходные закодированнные данные. Хотя это может показаться бесполезной информацией, мы покажем, что она позволяет противнику полностью восстановить исходное сообщение для любого выбранного шифртекста.

Для простоты, опишем атаку а трехблочный шифртекст. Пусть IV, c1, c2 — наблюдаемый атакующей стороной шифртекст, и пусть m1, m2 — исходные закодированные данные (неизвестные атакующему), соответствующие дополненному сообщению, как обсуждалось выше. (Каждый блок имеет длину L байтов.) Заметим, что

$$m2 = F_k^{-1}(c2) \oplus c1, \quad (3.15)$$

где k — это ключ (что, конечно, неизвесто атакующему), используемый честными участниками. Второй блок m2 заканчивается на $\underbrace{0\text{x}b \cdots 0\text{x}b}_{b \text{ times}}$, где

0xb обозначает однобайтную репрезентацию числа b. Ключевое свойство, используемое при этой атаке, заключается в том, что определенные изменения в шифртексте приводят к предсказуемым изменениям в исходных закодированных данных после расшифровки в режиме СШВ. А именно, пусть1 идентично c1 за исключением изменения в последнем байте. Рассмотрим расшифровку измененного шифртекста IV, c^r, c2. Ее результатом будут закодированные данные m^r, m^r, где $m^r = F^{-1}(c2) \oplus c^r$. Сравнивая с Уравнением (3.15), мы видим, что m^r будет идентично m2 за исключением изменения в последнем байте. (Значение m^r непредсказуемо, но это не повредит атаке.) Аналогично, если if cr равно c1 за исключением изменения в его i-м байте, то расшифровка IV, cr, c2 даст mr, mr, где mr равно m2 за исключением изменения в его i-м байте. Более обобщенно, если $c^r = c1 \oplus \Delta$ для любой последовательности Δ, то расшифровка IV, cr, c2 дает m^r, m^r, где $m^r = m2 \oplus \Delta$.

Таким образом, атакующая сторона имеет значительный контроль над последним блоком закодированных данных.

Для тренировки посмотрим, как противник может это использовать, чтобы узнать b — количество дополнения. (Это укажет на длину исходного сообщения.) Напомним, что после расшифровки, получатель смотрит на значение b последнего байта второго блока закодированных данных, и затем проверяет, что все последние b байтов имеют одно и то же значение. Атакующий сначала изменяет первый байт из c1 и посылает полученный шифртекст IV, cr, c2 получателю. Если расшифровка неудачна (т.е. получатель возвращает ошибку), то это значит, что получатель проверяет все L байтов из mr, и следовательно b = L! В противном случае, атакующий узнает, что b < L, и может повторить этот процесс со вторым байтом, и так далее. Крайний слева байт, для которого расшифровка была неудачной, точно указывает на крайний слева байт,

проверяемый получателем, то есть фактически указывает на b.

Зная b , атакующий может узнать все байты сообщения один за другим. Проиллюстрируем эту идею для последнего байта сообщения, который мы обозначим B. Атакующий знает, что m2 заканчивается на 0xB0xb • • • 0xb (с 0xb повторяющимся b раз), и хочет узнать B. Определим

$$\Delta_i \stackrel{\text{def}}{=} 0\text{x}00 \cdots 0\text{x}00\ 0\text{x}i\ \overbrace{0\text{x}(b+1) \cdots 0\text{x}(b+1)}^{b \text{ times}}$$
$$\oplus\ 0\text{x}00 \cdots 0\text{x}00\ 0\text{x}00\ \overbrace{0\text{x}b \cdots 0\text{x}b}^{b \text{ times}}$$

для $0 \le i < 28$; т.е., последние b + 1 байтов из Δi содержат число i (представленного в шестнадцатеричной системе), и затем значение (b + 1) ⊕ b (в шестнадцатеричной системе) повторяющееся b раз. Если атакующий посылает IV, c1 ⊕ Δi, c2 получателю, то после расшифровки в режиме СШВ полученные закодированные данные будут равны 0x(B ⊕ i)0x(b + 1) • • • 0x(b + 1) (с 0x(b + 1) повторяющимся b раз). Расшифровка будет неудачной, если не выполняется 0x(B ⊕ i) = 0x(b + 1). Атакующий всего лишь должен попробовать максимум 28 значений Δ0, . . . , Δ28 пока расшифровка не будет удачной для некоторого Δi, и в этот момент он может сделать вывод, что B = 0x(b + 1) ⊕ i. Мы предоставим полное описания того, как расширить эту атаку, чтобы узнать весь открытый текст, а не только последний блок, в качестве упражнения.

Атаки оракулу дополнения на CAPTCHA . CAPTCHA — это поврежденное изображение, скажем, английского слова, которое может легко прочитать человек, но не компьютер. CAPTCHA используются для проверки того, что с Интернет-страницей взаимодействует пользователь-человек, а не автоматическая программа. Например, CAPTCHA используются в сервисах электронной почты, чтобы спаммеры не могли проникнуть в тысячи аккаунтов и использовать их для рассылки спама.

Одной из возможных конфигураций CAPTCHA может быть вариант, когда это отдельный сервис работающий на независимом сервере. Чтобы посмотреть, как это работает, обозначим веб-сервер SW , сервер CAPTCHA SC , и пользователя U. Когда пользователь U загружает Интернет-страницу на сервере SW , может произойти следующее: SW шифрует произвольное английское слово w с помощью ключа k который изначально известен SW и SC , и посылает получившийся шифртекст пользователю (вместе с Интернет-страницей). U передает шифртекст SC , тот его дешифрует, получает w, и выдает поврежденное изображение w пользователю U. Наконец, U посылает слово w назад серверу SW для проверки. Интересно то, что SC расшифровывает любой шифртекст, полученный от U и выдаст ошибку «плохого дополнения», если расшифровки не будет удачной, как мы описывали ранее. В этом случае, U имеет возможность произвести атаку оракула добавления, как описано выше, и таким образом разгадать CAPTCHA (т.е. узнать слово w) авто-

матически *без любого человеческого вмешательства,* из-за чего CAPTCHA теряет эффективность. Хотя можно пребегнуть к специальным мерам (например, SC возвращает произвольное изображение вместо ошибки расшифровки), на самом деле необходимо использовать шифровальную схему, устойчивую к АВШ .

Упражнения

Докажите Предположение 3.6.

Докажите, что Определение 3.8 не удовлетворяется, если П может зашифровать сообщения произвольной длины, и противник не ограничен условием выдавать сообщения одинаковой длины в эксперименте $\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\Pi}$

> **Подсказка:** Пусть q(n) — полиномиальное максимальное значение длины шифртекста, когда П используется для шифрования единичного бита. Затем, рассмотрим противника, который выдает m0 ∈ {0, 1} и равномерное m1 ∈ {0, 1}$^{q(n)+2}$.

Скажем, что П = (Gen, Enc, Dec), такое что k ∈ {0, 1}^{n}, алгоритм Enc_k определен только для сообщений максимально длины A(n) (для некоторого полиномиального A). Постройте схему, удовлетворяющую Определение 3.8, даже когда противник не ограничен условием выдавать сообщения одинаковой длины в эксперименте $\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\Pi}$.

Докажите, что Определение 3.8 и 3.9 эквивалентны.

Пусть |tt(s)| = A(|s|) для некоторого A. Рассмотрим следующий эксперимент:

Эксперимент неразличимости PRG $\mathrm{PRG}_{A,G(n)}$:

(a) Выбирается единообразный бит b ∈ {0, 1}. Если b = 0, то выбрать единообразный r ∈ {0, 1}A(n); если b = 1, то выбрать единообразный s ∈ {0, 1}n и установить r := tt(s).

(b) Противник A получает r, и выдает бит b^r.

(c) Исход эксперимента является 1 , если b^r = b, и 0 в противном случае.

Предоставьте определение псевдослучайного генератора на основе этого эксперимента, и докажите, что ваше определение эквивалентно Определению 3.14. (То есть, покажите, что tt удовлетворяет ваше определение тогда и только тогда, когда оно удовлетворяет Определение 3.14.)

Пусть tt будет псевдослучайным генератором с коэффициентом расширения A(n) > 2n. В каждом из следующих случаев, скажите, обязательно ли ttr — псевдослучайный генератор. Если да, предоставьте доказательство. Если нет, предоставьте контрпример.

(a) Определите ttr(s) $\stackrel{\mathrm{def}}{=}$ tt(s1 • • • s|n/2J), где s = s1 • • • sn.

(b) Определите ttr(s) $\stackrel{\mathrm{def}}{=}$ tt .0|s|»s..

(c) Определите ttr(s) $\stackrel{\mathrm{def}}{=}$ tt(s) « tt(s + 1).

Докажите обратное Теореме 3.18. А именно, покажите, что если tt — не псевдослучайный генератор, то Конструкция 3.17 не имеет неотличимых шифрований при наличии перехватчика.

(a) Определите понятие неотличимости для шифрования нескольких различных сообщений, в котором схема не должна скрывать, зашифровано ли одно и то же сообщение дважды.

(b) Покажите, что Конструкция 3.17 не удовлетворяет ваше определение.

(c) Дайте конструкцию схемы детерминистского шифрования (без сохранения состояния), которая удовлетворяет ваше определение.

Докажите безусловно существование псевдослучайной функции F:

$\{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ с $A_{key}(n) = n$ и $A_{in}(n) = O(\log n)$.

Подсказка: Вычислите равномерную функцию с логарифмической длиной аргумента.

Пусть F — псевдослучайная функция, сохраняющая длину. Для следующих конструкций функции с ключом $F_r : \{0,1\}^n \times \{0,1\}^{n-1} \to \{0,1\}^{2n}$, укажите, является ли F_r псевдослучайной функцией с ключом. Если является, докажите это. Если нет, покажите атаку.

(a) $F_k^r(x) \stackrel{\text{def}}{=} F_k(0\|x) \| F_k(1\|x)$.

(b) $F_k^r(x) \stackrel{\text{def}}{=} F_k(0\|x) \| F_k(x\|1)$.

Предположим, что псевдослучайные функции существуют. Докажите, что существует схема шифрования, имеющая несколько неразличимых шифрований при наличии перехватчика (т.е. она удовлетворяет Определение 3.19), но она не является устойчивой против ABOT (т.е. она не удовлетворяет Определение 3.22).

> **Подсказка:** Эта схема не обязана быть «естественной». Вам нужно будет использовать тот факт, что при атаке с выбором открытого текста противник может выбирать запросы к оракулу шифрования адаптивно .

Пусть F — функция с ключом. Рассмотрим следующий эксперимент:

Эксперимент неразличимости PRF $\text{PRFA}_{A,F}(n)$:

(a) Выбирается единообразный бит $b \in \{0,1\}$. Если $b = 1$, то выберите единообразный $k \in \{0,1\}^n$.

(b) A получает на входе 1^n. Если $b = 0$, то A получает доступ к равномерной функции $f \in \text{Func}_n$. Если $b = 1$, то A вместо этого получает доступ к $F_k(\bullet)$.

(c) A выводит бит b'.

(d) Исход эксперимента является 1 , если $b' = b$, и 0 в противном случае.

Дайте определение псевдослучайным функциям в этом эксперименте и до-

кажите, что ваше определение эквивалентно Определению 3.25.

Рассмотрим функцию с ключом F : Для параметра безопасности n, ключ — это матрица A логических значений размера $n \times n$ и n-битный вектор логических значений b. Дайте определение $F_{A,b} : \{0,1\}^n \to \{0,1\}^n$ by $F_{A,b}(x) \stackrel{def}{=} Ax + b$, где все операции производятся по модулю 2. Продемонстрируйте, что F не является всевдослучайной функцией .

Докажите, что если F — псевдослучайная функция, сохраняющая длину, то $tt(s) \stackrel{def}{=} F(1)\|F(2)\|\bullet\bullet\bullet\|F(A)$ — псевдослучайный генератор с коэффициентом расширения $A \bullet n$.

Определите понятия абсолютной стойкости при атаке с выбором открытого текста, изменив Определение 3.22. Продемонстрируйте, что определение не может быть завершено.

Докажите Предположение 3.27.

Подсказка: Используйте результаты Приложения A.4.

Предположим существование псевдослучайных перестановок. Покажите, что существует функция F^r, которая является псевдослучайной перестановкой, но не является строгой псевдослучайной перестановкой.

Подсказка: Постройте F^r, такую что $F^r(k) = 0|k|$.

Пусть F — псевдослучайная перестановка. Определим шифровальную схему с фиксированной длиной (Enc, Dec) следующим образов: На входе $m \in \{0,1\}^{n/2}$ и ключ $k \in \{0,1\}^n$, алгоритм Enc выбирает единообразную последовательностьr $\in \{0,1\}^{n/2}$ длины n/2 и вычисляет $c := Fk(r\|m)$.

Покажите расшифровку и докажите, что эта схема устойчива к АВОТ для сообщений длины n/2. (Если вы хотите действительно сложное задание, докажите, что эта схема устойчива к АВШ, если F — строгая псевдослучайная перестановка.)

Пусть F — псевдослучайная функция и tt — псевдослучайный генератор с коэффициентом расширения $A(n) = n + 1$. Для каждой из следующий схем шифрования укажите, обладает ли схема неразличимыми шифрованиями в присутствии перехватчика и устойчива ли она к АВОТ. (В каждом случае общедоступным ключом является единообразный $k \in \{0,1\}^n$.) Поясните свой ответ.

(a) Для шифрования $m \in \{0,1\}^{n+1}$, выбрать идинообразную $r \in \{0,1\}^n$ и выдать шифртекст $(r, tt(r) \oplus m)$.

(b) Для шифрования $m \in \{0,1\}n$, выдать шифртекст $m \oplus Fk(0n)$.

(c) Для шифрования $m \in \{0,1\}2n$, анализировать m как m1»m2 с $|m1| = |m2|$, затем выбрать единообразную $r \in \{0,1\}n$ и послать $(r, m1 \oplus Fk(r), m2 \oplus Fk(r+1))$.

Рассмотрим версию шифрования в режиме СШВ с сохранением состояния,

где отправитель просто увеличивает IV на 1 каждый раз, когда шифруется сообщение (вместо того, чтобы произвольно выбирать IV каждый раз). Покажите, что полученная схема не устойчива к АВОТ.

Каков эффект однобитной ошибки в шифртексте при использовании режимов СШВ, ОСВ и режима счетчика?

Каков эффект произведет пропущенный блок шифртекста (т.е. если передаваемый шифртекст c1, c2, c3, . . . получен как c1, c3, . . .) при использовании режимов СШВ, ОСВ и режима счетчика ?

Представим, что для шифрования сообщения 1024-битной длины используется шифрование в режиме СШВ используется с блочным шифром, имеющим 256-битный ключ и 128-битную длину блока. Какова длина полученного шифртекста?

Предоставьте подробности доказательства от обратного для Уравнения (3.12).

Пусть F — псевдослучайная функция, такая что для $k \in \{0,1\}^n$ функция Fk отображает Ain(n)-битные входные данные на Aout(n)-битные выходные данные.

(a) Рассмотрим применение шифрования в режиме счетчика с помощью F . Для каких функций Ain, Aout полученная схема шифрования устойчива к АВОТ?

(b) Рассмотрим применение шифрования в режиме счетчика с помощью F , но только для сообщений фиксированной длины A(n) (которое является целым кратным Aout(n)). Для каких Ain, Aout, A эта схема имеет неразличимые шифрования в присутствии перехватчика?

Для любой функции $g : \{0,1\}^n \to \{0,1\}^n$, определим g$(•) как вероятностн ый оракул, который на входе 1^n, выбирает единообразную $r \in \{0,1\}^n$ и возвращает (r, g(r)). Функция с ключом F является слабой псевдослучайной функцией для всех ppt алгоритмов D, существует пренебрежимо малая функция negl, такая что:

$$\left| \Pr[D^{F_k^{\$}(\cdot)}(1^n) = 1] - \Pr[D^{f^{\$}(\cdot)}(1^n) = 1] \right| \leq \mathsf{negl}(n),$$

где $k \in \{0,1\}^n$ и $f \in \mathrm{Func}_n$ выбраны равномерно.

(a) Докажите, что если F псевдослучайна, то она слабо псевдослучайна.

(b) Пусть F r — псевдослучайная функция. Определим

$$F_k(x) \stackrel{\text{def}}{=} \begin{cases} F'_k(x) & \text{if } x \text{ is even} \\ F'_k(x+1) & \text{if } x \text{ is odd.} \end{cases}$$

Докажите, что F слабо псевдослучайна, но не псевдослучайна.

(c) Обязательно ли шифрование в режиме счетчика, использующее слабо псевдослучайную функцию, является учтойчивым в АВОТ? Обязательно ли оно имеет неразличимые шифрования в присутствии перехватчика? Докажите

свои ответы.

(d) Докажите, что Конструкция 3.30 устойчива к АВОТ, если F — слабо псевдослучайная функция.

Пусть F будет псевдослучайной перестановкой. Рассмотрим режим, при котором выбирается единнобразное значение $ctr \in \{0, 1\}^n$, и i-ый блок шифртекста ci вычисляется как $ci := Fk (ctr + i + mi)$. Покажите, что эта схема не имеет неразличимых шифрований в присутствии перехватчика.

Покажите, что режимы СШВ, ОСВ и режим счетчика не производят схем, устойчивых в АВШ (независимо от F).

Пусть $П1 = (Enc1, Dec1)$ и $П2 = (Enc2, Dec2)$ — две шифровальных схемы, для которых известно, что по крайней мере одна из них устойчива к АВОТ (но мы не знаем, кака именно). Покажите, как построить шифровальную схему П, которая точно устойчива к АВОТ, если по крайней мере П1 или П2 устойчива к АВОТ. Приведите полное доказательство своего решения.

> **Подсказка:** Создайте два сообщения с открытым текстом из исходного открытого текста, так чтобы их знание ничего не раскрывало бы об исходном открытом тексте, а знание обоих позволяло бы вычислить исходный открытый текст.

Напишите псевдокод для нахождения всего открытого текста через атаку оракула дополнения на шифрование в режиме СШВ с помощью дополнения PKCS #5, как описано в тексте.

Опишите атаку оракула дополнения на шифрование в режиме счетчика (предполагая, что для дополнения длины сообщений до длины, кратной блочной длине, применялось дополнение PKCS #5 до шифрования).

Глава 4

Коды аутентификации сообщений

4.1 Целостность сообщения

4.1.1 Секретность и целостность

Одной из самых основных целей криптографии является предоставление возможности безопасной коммуникации через *открытый канал связи*. Но что означает «безопасная коммуникация»? В Главе 3 мы показали, что возможно засекреченная коммуникация через открытый канал. То есть мы показали, как шифрование может быть использовано для того, чтобы помешать перехватчику (или более активному противнику) узнать что-либо о содержании сообщений, посылаемых через незащищенных канад связи. Однако, не все проблемы безопасности связаны с секреттностью. Часто такой же или даже более важной задачей является обеспечение целостности сообщения (или аутентификации сообщения) в том смысле, что каждый участник должен иметь возможность определить, когда полученное сообщение было послано участником, который утверждает что его послал именно он, и что сообщение не было изменено при передаче. Рассмотрим два канонических примера.

Представим пользователя, который связывается со своим банком через Интернет. Когда банк получает запрос на перевод $1,000 со счета пользователя на счет другого пользователя X, банк должен ответить на следующие вопросы:

1. Является ли запрос настоящим? То есть, действительно ли данный пользователь послал этот запрос, или запрос был послан противником (возможно, самим пользователем X), выдающим себя за законного пользователя?

2. Предполагая, что запрос на перевод был послан законным пользователем, являются ли детали запроса, в том виде, в каком они получены, такими как предполагал законный пользователь? Или, например, была изменена сумма перевода?

Заметим, что обычные методы коррекции ошибок недостаточны для ответа на второй вопрос. Коды коррекции ошибок обнаруживают и исправляют только «произвольные» ошибки, затрагивающие только малую часть передачи, они не защищают от злонамеренного противника, который может выбрать, где внедрить любое количество ошибок.

Второй случай, в котором возникает необходимость в целостности сообщения, — это случай веб-куки. Протокол HTTP, который используется в передаче данных в сети, не сохраняет состояния, так что когда между клиентом и сервером происхо-

дит сеанс связи (например, когда пользователь [клиент] покупает на сайте продавца [сервера]), любое состояние, созданное во время этого сеанса (например, содержание корзины пользователя) часто помещается в «куки», которое хранится клиентом и посылается от клиента к серверу как часть любого сообщения, посылаемого клиентом. Предположим, что куки, хранимое некоторым пользователем, включает предметы в корзине пользователя вместе с ценами на каждый предмет, как если продавец предлагает разные цены разным клиентам (например, отображая скидки и специальные предложения, или специальные цены для отдельных пользователей). Пользователь не должен иметь возможность изменить куки, которое он хранит, чтобы изменить цены на предметы в своей корзине. Таким образом, продавцу нужен метод обеспечения целостности куки, хранимого у пользователя. Заметим, что содержание куки (а именно предметы и их цены) не засекречены и даже должны быть известны пользователю. Таким образом, проблема заключается именно в целостности.

На самом деле, нельзя считать коммуникацию целостной, если не было предпринято никаких мер для обеспечения этой целостности. Действительно, нельзя с точностью утверждать ни о каком незащищенной заказе товара, банковской операции онлайн, электронном письме или СМС сообщении, что они были отправлены из заявленного источника, и что они не были изменены при передаче. К сожалению, люди часто доверчивы, так что такая информация как номер вызывающего абонента или адрес электронной почты рассматриваются как «доказательство источника», хотя их достаточно легко фальсифицировать. Это создает возможность для потенциально опасных атак.

В этой главе мы покажем, как обеспечить целостность сообщений с использованием методов криптографии, чтобы предотвратить необнаруженное повреждение сообщений в открытом канале связи. Заметим, что мы не можем предотвратить все повреждения сообщений противником в целом, так как от этого можно защитить только на физическом уровне. Вместо этого, мы гарантируем, что любое такое повреждение будет *обнаружено* честными участниками.

4.1.2 Шифрование и аутентификация сообщений

Цели секретности и целостности сообщений различны, и методы их достижения также различны. К сожалению, секретность и целостность часто путаются и ненужным образом объединяются, так что сразу разъясним: шифрование (в общем) не обеспечивает целостность, и шифрование никогда не должно использоваться для аутентификации сообщений, если оно не создано специально для этой цели (мы вернемся к этому в Разделе 4.5).

Можно ложно предположить, что шифрование решает проблему аутентификации сообщений. (Это даже является популярной ошибкой.) Это происходит из-за запутанных, ложных заключений о том, что так как шифртекст полностью

скрывает содержание сообщения, противник не может изменить зашифрованное сообщение значимым образом. Несмотря на интуитивную привлекательность этого заключения, оно абсолютно ложное. Мы продемонстрируем это, показав, что ни одна из схем шифрования, рассмотренных нами до сих пор, не обеспечивает целостность сообщения.

Шифрование с помощью поточных шифров . Рассмотрим простую шифровальную схему, в которой Enck (m) вычисляет шифртекст c := tt(k) ⊕m, где tt — псевдослучайный генератор. Шифртексты в этом случае могут быть очень легко манипулируемы: переключение любого бита в шифртексте c приведет к переключению того же бита в сообщении, полученном при дешифровке. Так, зная шифртекст c который шифрует (возможно неизвестное) сообщение m, можно создать модифицированный шифртекст c^r, такой что mr := Deck (c^r) равно m, но с одним (или более одного) переключенным битом. Эта простая атака может иметь тяжелые последствия. В качестве примера, рассмотрим случай, когда пользователь шифрует некоторое количество долларов, которое он хочет перевести со своего банковского счета, где эта сумма представлена в двоичной системе исчисления.

Переключение самого незначительно бита приводит к изменению суммы всего на $1, но переключение 11-го незначительного бита изменяет сумму на более чем $1000! (Интересно, что противник в этом примере не знает, уменьшается или увеличивается ли исходная сумма, т.е. происходит ли переключение с 0 на 1 или наоборот. Но если противник обладает частичными знаниями о сумме, например, для начала, что она меньше $1000, то внесенные им изменения могу иметь предсказуемый эффект.) Обратим внимание, что эта атака не противоречит секретности шифровальной схемы (в значении Определения 3.8). На самом деле, эта же самая атака применима к шифровальной схеме Вермана, что показывает, что даже полная секретность недостаточна даже для самого базового уровня целостности.

Шифрование с помощью блочных шифров . Атака, описанная выше, использует тот факт, что переключение единичного бита в шифртексте не меняет исходный открытый текст, за исключением соответствующего бита (который тоже переключается). Эта же атака применима к шифровальным схемам в режиме ОСВ и режиме счетчика, которые тоже шифруют сообщения, объединяя их с помощью исключающего ИЛИ с псевдослучайным потоком (хотя он и меняется при каждой шифровке сообщения). Таким образом, мы видим, что даже шифрования, устойчивого к АВОТ, недостаточно для предотвращения повреждения сообщений.

Можно надеяться, что атаковать шифрование в режиме КЭК или СШВ намного сложнее, так как расшифровка в этих случаях использует (строгую) псевдослучайную перестановку F_k , и мы ожидаем, что F_k −1(x) и F −1(xr) не будут коррелировать, даже если x и xr отличаются только на один бит.

(Конечно, режим КЭК не гарантирует даже самый базовый уровень секретности, но это неважно для данного обсуждения.) Однако, изменения одного бита в шифртексте все равно приводят к предсказуемым изменениям в открытом тексте. Например, при использовании режима КЭК, переключение бита в i-м блоке шифртекста влияет только на i-й блок открытого текста, а все остальные блоки не меняются. Хотя предсказать эффект, оказанный на i-й блок открытого текста может быть невозможно, изменение этого одного блока (оставляя все остальное неизменным) может представлять собой опасную атаку. Кроме того, порядок блоков открытого текста может быть изменен (без искажения блоков), просто поменяв порядок соответствующих блоков шифртекста, а сообщение может быть сокращено, просто отбросив блоки шифртекста.

Аналогично, при использовании режима СШВ, переключение j-ого бита в IV меняет только j-й бит в первом блоке сообщения m1 (так как $m1 := F^{-1}(c1) \oplus IV r$, где IV r —это измененный IV). Все блоки шифртекста помимо первого остаются неизменными (так как i-ый блок открытого текста вычисляется как $mi := F^{-1}(ci) \oplus ci$, и блоки ci и ci−1 не менялись). Таким образом, первый блок сообщения, зашифрованного в СШВ, можно изменить произвольно. На практике, это важная проблема, так как первый блок часто содержит важную информацию из заголовка. Наконец, заметим, что все шифровальные схемы, рассмотренные нами до сих пор, имеют свойство, что каждый шифртекст (возможно, ограниченный определенной длиной) соответствует какому-либо сообщению. Так что для любого противника очень просто послать ложное сообщение от лица одного из участников коммуникации, послав какой-либо произвольный шифртекст, даже если противник не знает, каким будет исходное сообщение. Как мы увидим при формальном доказательстве аутентифицированного шифрования в Разделе 4.5, даже такого рода атака должна быть предусмотрена.

4.2 Коды аутентификации сообщений – определения

Мы удостоверились что, в общем смысле, шифрование не решает проблему целостности сообщения. Скорее необходим дополнительный механизм, который позволит участникам коммуникации понять, было ли сообщение намеренно повреждено. Подходящим инструментом для этой задачи является код аутентификации сообщения (КАС).

Цель кода аутентификации сообщения состоит в том, чтобы не позволить противнику изменить сообщение, посланное от одного участника к другому, или ввести новое сообщение без того, чтобы получатель установил, что сообщение не исходит от правильного участника. Как в случае шифрования, это возможно только в том случае, когда участники коммуникации имеют общий секрет, который не известен противнику (иначе противник может выдать себя на отправителя). Мы снова рассмотрим сценарий, когда имеется закрытый ключ, который известен участникам.1

Синтаксис кода аутентификации сообщений

Прежде чем дать формальное определение устойчивости кода аутентификации сообщений, определим сначала, что такое КАС и как он используется. Два пользователя, желающие общаться аутентифицированным образом, начинают с того, что делятся секретным ключом k перед коммуникацией. Когда один из участников хочет послать сообщение m другому, он вычисляет тэг КАС (или просто тэг) t на основе сообщения и общего ключа, и посылает сообщение m и тэг t другому участнику. Тэг вычисляется с помощью алгоритма генерации тэга, обозначенного Mac. Таким образом, иными словами, отправитель сообщения m вычисляет $t \leftarrow$ Mack (m) и передает (m, t) получателю. По получении (m, t), второй участник проверяет, является ли t действующим тэгом для сообщения m (в отношении общего ключа) или нет. Это делается путем запуска алгоритма верификации Vrfy , который принимает на входе общий ключ, а также сообщение m и тэг t, и указывает, является ли данный тэг действующим. Формально:

ОПРЕДЕЛЕНИЕ 4.1 Код аутентификации сообщения (или КАС) *состоит из трех вероятностных алгоритма, работающих за полиномиальное время* (Gen, Mac, Vrfy) , *таких как* :

4.2.2 Алгоритм генерации ключа Gen *принимает на входе параметр безопасности 1n и выдает ключ k , при* $|k| \geq n$.

4.2.3 Алгоритм генерации тэга Mac *принимает на входе ключ k и сообщение* $m \in \{0, 1\}^*$, *и выдает тэг t. Так как этот алгоритм может быть рандомизирован, мы запишем это как* $t \leftarrow Mack (m)$.

4.2.4 Детерминистский алгоритм верификации Vrfy *принимает на входе ключ k , сообщение m, и тэг t. Он выдает бит b, при этом b = 1 означает действителен, b = 0 означает недействителен. Запишем это как* $b := Vrfyk (m, t)$.

Необходимо, чтобы для каждого n каждый ключ k, выведенный алгоритмом $Gen(1^n)$ *и каждого* $m \in \{0, 1\}^*$, *было бы верным утверждение* $Vrfyk (m, Mack (m)) = 1$.

Если существует функция A , такая что для каждого k , полученного Gen(1n), алгоритм Mack определен только для сообщений $m \in \{0, 1\}A(n)$, *то мы назовем схему КАС с фиксированной длиной для сообщений длиной A(n).*

Как и при шифровании с закрытым ключом, $Gen(1^n)$ почти всегда просто выбирает единообразный ключ $k \in \{0, 1\}^n$, и в таком случае мы опускаем Gen .

Каноническая верификация . Для детерминистских кодов аутентификации

---

[1]В примере web-cookie , рассмотренном выше, продавец (на самом деле) общается «сам с собой», в то время как пользователь выступает как канал коммуникации. При такой расстановке, только сервер должен знать ключ, так он выступает как отправитель и получатель.

сообщений (то есть когда Mac — детерминистский алгоритм), канонический метод верификаци и заключается в том, чтобы просто заново вычислить тэг и проверить равенство двух тэгов. Иными словами, Vrfyk (m, t) сначала вычисляет t˜ := Mack(m), а потом выдает 1 тогда и только тогда, когда t˜ = t. Однако, даже для детерминистских КАСов, полезно определить отдельный алгоритм Vrfy , чтобы четко различать семантику аутентификации сообщения и проверку его подлинности.

Безопасность кодов аутентификации сообщений

Определим теперь исходное понятие безопасности для кодов аутентификации сообщений. Интуитивная идея, лежащая в основе этого определения, следующая: ни один эффективный противник не может сгенерировать действительный тэг для любого «нового» сообщения, которое ранее не посылалось (и не было аутентифицировано) никем из участников коммуникации.

Как в любом определении безопасности, для формализации этого понятия нам нужно определить как силу противника, так и что именно будет пониматься под «взломом». Как обычно , мы рассматриваем только вероятностных противников, работающих в полиномиальном времени[2] и настоящий вопрос заключается в том, как создать модель взаимодействия противника с участниками коммуникации. При аутентификации сообщения противник, следящий за коммуникацией между честными участниками, может видеть все сообщения, посылаемые участниками, вместе с их соответствующими тэгами КАС. Противник также может повлиять на содержание этих сообщений, напрямую или косвенно (если, например, внешние действия противника влияют на сообщения участников). Это применимо, например, к примеру сетевых куки, где действия самого пользователя влияют на содержание куки, сохраняемого на его компьютере.

Для создания соответствующей модели мы позволяем противнику запросить тэги КАС для любых выбранных им сообщений. Формально, мы даем противнику доступ к оракулу КАСа Mack (•). Противник может повторно послать любое сообщение m этому оракулу, и взамен получает тэг t ← Mack (m). (Для всех КАСов фиксированной длины могут посылаться только сообщения соответствующей длины.)

Мы будем считать «взломом» схемы, если противник может выдать любое сообщение m вместе с тэгом t , такое что: (1) t — действующий тэг для сообщения m (т.е., Vrfyk (m, t) = 1), и (2) противник ранее не запрашивал тэг КАС для сообщения m (т.е. из его оракула). Первое условие означает, что если бы противник хотел послать (m, t) одному из честны х участников, то этот участник бы

---

[2]См. Раздел 4.6, где обсуждается информационно-теоретическая аутентификация сообщений, где на противника не накладывается никаких вычислительных ограничений.

ложно подумал, что m послано легитимным участником, так как Vrfyk (m, t) = 1. Второе условие требуется, потому что противник всегда может просто скопировать сообщение и тэг КАС, посланные ранее одним из легитимных участников (и, конечно, они будут приняты как действующие). Такая *повторная атака* не считается «взломом» кода аутентификации сообщения. Это не означает, что повторные атаки не представляют проблему для безопасности. Они представляют, и мы вернемся к этому позже.

КАС, соответствующий уровню безопасности, описанному выше, называют *экзистенциально не поддающийся подделке при адаптивной атаке с выбором сообщения.* «Невозможность пооделки» означает, что противник не должен иметь возможность подделать действующий тэг для любого сообщения, а «адаптивная атаки с выбором сообщения» означает, что противник может получить тэги КАС для произвольных сообщений, выбранных адаптивно во время атаки.

Для формального определения, рассмотрим следющий эксперимент для кода унтентификации сообщений Π = (Gen, Mac, Vrfy), противника A, и значение n для параметра безопасности:

Эксперимент аутентификации сообщения Mac-forge

1. С помощью алгоритма Gen(1^n) генерируется ключ k.

2. Противник A получает вход 1n и доступ к Mack (•). Противник в конечном итоге выдеает (m, t). Пусть Q обозначает множество всех запросов, которые A задал своему оракулу.

3. A умеет успех тогда и только тогда, когда (1) Vrfyk (m, t) = 1 и (2) m f ∈ Q. *В этом случае исход эксперимента будет 1.*

КАС является безопасным, если никакой эффективный противник не может иметь успех в этом эксперименте с непренебрежимой вероятностью:

ОПРЕДЕЛЕНИЕ 4.2 Код аутентификации сообщения Π = (Gen, Mac, Vrfy) экзистенциально не подлежит подделке при адаптивной атаке с выбором сообщения, или просто безопасен, если для всех вероятностных противников, работающих в полиномиальном времени A, существует пренебрежимо малая функция negl , такая что:

$\Pr[\text{Mac-forge}_{A\Pi}(n) = 1] \leq \text{negl}(n)$.

Является ли это определение слишком строгим? Вышеприведенное определение достаточно строгое в двух отношениях. Во-первых, противнику разрешается запросить тэги КАС для любых сообщений по своему выбору. Во-вторых, считается, что противник «взломал» схему, если он может выдать действующий тэг для любого сообщения, прежде не аутентифицированного. Можно поспорить, что оба эти компонента определения нереалистичны и слишком строги: при использовании КАС в «реальном мире» честные участники аутентифицируют только «значимые» сообщения (контроль противника над которыми ограничен),

и соответственно, если противник может подделать действующий тэг для «значимого» сообщения, то это должно считаться только нарушением безопасности. Почему мы не можем исправить определение, чтобы это учесть?

Важнейшим моментом здесь является то, что понятие значимого сообщения полностью *зависит от приложения*. Хотя некоторые приложения КАСа могут аутентифицировать только сообщения на английском языке, другие приложения могут аутентифицировать файлы с электронными таблицами, базы данных и другие необработанные данные. Также можно создавать протоколы, где может быть аутентифицировано все, что угодно — в действительности, некоторые протоколы для аутентификации объектов как раз это позволяют. Делая определение безопасности для КАС как можно строже, мы удостоверяемся, что безопасные КАС широко применимы для многих целей, без необходимости беспокоиться о совместимости КАС с семантикой приложения.

Повторные атаки. Подчеркнем, что вышеприведенное определение, как и коды аутентификации сообщений в отдельности, не обеспечивают защиты против повторных атак, когда ранее посланное сообщение (и его тэг КАС) передаются честному участнику. Несмотря на это, повторные атаки представляют серьезную проблему! Рассм отрим снова сценарий, где пользователь (скажем, Элис) посылает запрос своему банку для перевода $1,000 с ее счета другому пользователю (скажем, Бобу). Для этого Элис может вычислить тэг КАС и добавить его к своему запросу, чтобы банк знал, что запрос достоверен. Если КАС безопасен, Боб не сможет перехватить запрос и изменить сумму на $10 000, потому что для этого потребовалось бы подделать действующий тэг для ранее не аутентифицированного сообщения. Однако, ничто не мешает Бобу перехватить сообщение Элис и повторить его банку десять раз. Если банк примет каждое из этих повторных сообщений, в результате на счет Боба будет переведено $10 000 вместо желаемых $1000. Несмотря на реальную угрозу, представляемую повторными атаками, один только КАС не может защитить от таких атак, так как определение КАС (Определение 4.1) не включает никого понятия о состоянии в алгоритм верификации (поэтому каждый раз, когда алгоритму верификации дана действующая пара (m, t), он всегда выдаст 1). Вместо этого, защита от повторных атак, если такая защита вообще необходима, должна осуществляться каким-либо приложением высшего уровня. Причина, по которой определение КАСа построено таким образом, заключается в том, что мы не хотим предполагать никакой семантики в отношении приложений, использующих КАС. В частности, решение о том, должно ли повторное сообщение считаться «действующим» может зависеть от приложения.

Есть два метода предотвращения повторных атак: использовать *последовательные числа* (также известные как счетчики) или временные штампы). Первый подход, который еще раз будет описан (в более широком контексте) в Разделе 4.5.3, требует, чтобы участники коммуникации сохраняли (синхронизованное) состоя-

ние, что может быть сложно, если пользователи общаются через канал с потерями, в котором сообщения иногда теряются (хотя эту проблему можно уменьшить). При использовании второго метода, временных штампов, отправитель добавляет к сообщению текущее время T (скажем, ближайшую миллисекунду) перед аутентификацией, и посылает T вместе с сообщением и с полученным тэгом t. Когда получатель получает T, m, t, он проверяет, что t действителен для T «m и что T находится в пределах приемлемого временного отклонения от текущего времени T^r при получении. У этого метода также есть определенные недостатки, например, необходимо, чтобы отправитель и получатель имели тесно синхронизованные часы, а также возможность осуществить повторную атаку, если это сделать достаточно быстро (а именно, в пределах приемлемого времменного окна).

Строгие КАСы. По определению, безопасный КАС предотвращает возможность противника генерировать действующий тэг для нового сообщения, которое ранне не было аутентифицаровано. Но он не предотвращает возможность того, что атакующий может сгенерировать новый тэг для ранее аутентифицированного сообщения. То есть, КАС гарантирует , что если атакующий знает тэги t1, . . . для сообщений m1, . . ., то он не может подделать действующий тэг t для любого сообщения m $f\in$ {m1, . . .}. Однако, противник может «подделать» другой действующий тэг tr f= t1 для сообщения m1. В целом, такой тип поведения противника не представляет проблем.

Несмотря на это, в некоторых ситуациях полезно использовать более строгое определение безопасности КАСов, в котором учитывается такое поведение.

Формально, рассмотрим модифицированный эксперимент Mac-sforge , который определен точно так же, как Mac-forge, за исключением того, что теперь множество Q содержит пары запросов к оракулу и соответствующих им ответов. (То есть, (m, t) ∈ Q если A послал запрос Mack (m) и получил в ответ тэг t.) Противник A будет иметь успех (и эксперимент Mac-sforge даст 1) тогда и только тогда, когда A выдаст (m, t), такие что Vrfyk (m, t) = 1 и (m, t) ∈/ Q.

ОПРЕДЕЛЕНИЕ 4.3 *Код аутентификации сообщения* Π = (Gen, Mac, Vrfy) *строго безопасен или называется сильным* КАСом*, если для всех вероятностных противников, работающих в полиномиальном времени A, существует пренебрежимо малая функция* negl *, такая что*:

$\Pr[\text{Mac-sforge}_{A,\Pi}(n) = 1] \leq \text{negl}(n)$.

Не сложно увидеть, что безопасный КАС использует каноническую верификацию, то он также строго безопасен. Это важно, потому что все реальные КАСы используют каноническую верификац ию. Мы оставим доказательство следующего утверждения для упражнения.

ПРЕДПОЛОЖЕНИЕ 4.4 *Пусть* Π = (Gen, Mac, Vrfy) — *безопасный* КАС, *использующий каноническую верификацию. Тогда* Π — *сильный* КАС.

Верификационные запросы

Определения 4.2 и 4.3 дают противнику доступ к оракулу КАСа, что соответствует реальному противнику, способному повлиять на честного участника, чтобы он сгенерировал тэг для некоторого сообщения m. Также можно рассмотреть противника, который взаимодействует с честным получателем, посылая mr, tr получателю, чтобы узнать, верно ли, что Vrfyk (m^r , t^r) = 1. Такой противник может быть пойман формально естественным путем, если в определениях, рассмотренных выше, дать противнику также доступ к орак улу верификации.

Определение, которое бы включало таким образом оракул верификации, возможно, было бы «правильным» определением для безопасности кодов аутентификации сообщений. Однако оказывается, что КАСов, использующих каноническую верификацию, нет никакой разницы: любой КАС , соответствующий Определению 4.2 также соответствует вариант определения, в котором разрешены верификационные запросы. Аналогично, любой сильный КАС автоматически остается безопасным в ситуации, когда возможны верификационные запросы. (Кстати, это служит мотивацией для определения строгой безопасности для КАСов.) Однако для КАСов, которые не используют каноническую верификацию, разрешение верификационных запросов имеет определенные последствия. См. Упрежнения 4.2 и 4.3. Так как большинство КАСов в этой книге (так же как и КАСы, используемый на практике), используют каноническую верификацию, мы используем традиционные определения, которые не упоминают доступ к оракулу верификации.

Потенциальная временная атака . Одна из проблем, которой еще не уделялось внимание, заключается возможности провести временную атаку на верификацию КАСа. Здесь действует противник, посылающий пары сообщений и тэгов получателю, используя его таким образов как оракул, и узнает не только принимает ли получатель или отклоняет, но также и время, необходимое получателю для принятия этого решения. Мы показываем, что если такая атака возможна, то естественная имплементация верификации КАСов приведет к уязвимости, которую можно легко использовать. Заметим, что в наших обычных определениях безопасности атакующий узнает только результат работы оракулов, к которым он имеет доступ, и ничего более. Описываемая здесь атака, которая является примером *атаки на побочный канал,* показывает, что некоторые реальные атаки не описываются обычными определениями.)

Более конкретно, рассмотрим КАС, исполь зующий каноническую верификацию. Чтобы проверить тэг t для сообщения m, получатель вычисляет t^r := Mack (m), а затем вычисляет t^r to t, выдавая 1 тогда и только тогда, когда tr и t равны. Предположим, что это сравнение производится стандартным образом (как strcmp в C), сравнивая t and tr по одному байту и давая отказ, как только найден первый неравный байт. Наблюдение состоит в том, что при таком подходе время, потраченное на отказ, может быть разным в зависимости от позиции первого неравного байта.

Эту с виду иррелевантную информацию можно использовать для подделки любого желаемого сообщения m. Основная идея такова: предположим, что атакующий знает первые i байтов правильного тэга для m. (В начале i = 0.) Атакующий узнает следующий байт правильного тэга, послав получателю (m, t0), . . . , (m, t255), где tj — это последовательность первых i правильных байтов, (i + 1)-ый байт равен j (в шестнадцатеричной системе исчисления), а остальные байты заданы как 0x00. Все эти тэги, скорее всего, получат отказ (если нет, то атакующий все равно будет иметь успех). Однако, для ровно одного из этих тэгов первые (i + 1) байтов совпадут с правильным тэгом, и отказ займет немного больше времени, чем для других. Если тэг tj имел самое долгое время отказа, атакующий узнает, что (i + 1)-ый байт правильного тэга равен j. Таким образом, атакующий узнает каждый байт правильного тэга, испольщуя максимум 256 запросов к оракулу верификации. Для 16-байтного тэга такая атака потребует только 4096 запросов в худшем случае.

Можно засомневаться в том, что такая атака реалистична, так как она требует доступ к оракулу верификации, а также способность измерять разницу между во времени, затраченном на сравнение i и i + 1 байтов. В действительности, именно такие атаки были произведены на реальные системы! Один только пример: КАСы были использованы для проверки обновлений кода в Xbox 360, и использованная там имплементация верификации КАСов имела разницу во времени отказа 2,2 миллисекунды. Атакующие смогли воспользоваться этим и загрузить в оборудование пиратированные игры.

На основе вышеупомянутого, мы делаем вывод, что верификация КАСов должна использовать сравнение последовательностей, *не зависимое от времени*, которое всегда сравнивает все байты.

4.3 Построение безопасных кодов аутентификации сообщений

4.3.1 КАС с фиксированной длиной

Естевственным инструментом для построения безопасных кодов аутентификации сообщений являются псевдослучайные функции. Интуитивно, если тэг КАСа t получен путем применения псевдослучайной функции к сообщению m, то для подделки тэга для ранее не аутентифицированного сообщения противник должен правильно угадать значение псевдослучайной функции на «новых» входных данных. В ероятность правильного угадывания значения случайной функции на новых входных данных равна 2–n (если длина выходных данных функции равна n) . Вероятность правильного угадывания такого значения для п севдослучайной функции может быть только пренебрежимо больше.

Эта идея, показанная в Конструкции 4.5, работает для построения безопасного КАСа фиксированной длины для сообщений длины n (так как наши псевдослучайные функции имеют длину блока, по умолчанию равную n битам). Эт о полезно, но недостаточно для нашей цели. В Разделе 4.3.2 мы покажем, как расширить эту идею для работы с сообщениями произвольной длины. Мы изучим

более эффективные методы построения КАСов для сообщений произвольной длины в Разделах 4.4 и 5.3.2.

КОНСТРУКЦИЯ 4.5

Пусть F — псевдослучайная функция. Определим КАС фиксированной длины для сообщений длины n следующим образом:

• Mac: при вводе ключа $k \in \{0,1\}$ and a message $m \in \{0,1\}$, вывести тэг $t := Fk(m)$. (Если $|m| \neq |k|$, то не выдавать ничего.)

• Vrfy: при вводе ключа $k \in \{0,1\}$, сообщения $m \in \{0,1\}$, и тэга $t \in \{0,1\}^n$, вывести 1 тогда и только тогда, когда $t = F(m)$. (Если $|m| \neq |k|$, то вывести 0.)

КАС фиксированной длины из любой псевдослучайной функции.

ТЕОРЕМА 4.6 *Если F — псевдослучайная функция, то Конструкция 4.5 является* КАСом *с фиксированной длиной для сообщений длины n.*

ДОКАЗАТЕЛЬСТВО Как и в предыдущих случаях использования псевдослучайных функций, это доказательство следует парадигме, в которой сначала анализируется устойчивость схемы, используя действительно случайную функцию, а затем рассматривается результат замены действительно случайной функции на псевдослучайную.

Пусть A – вероятностный полиномиально-временной противник. Рассмотри код аутентификации сообщения $\Pi\tilde{} = (G\tilde{}en, M\tilde{}ac, V\tilde{}rfy)$, который равен $\Pi = (Mac, Vrfy)$ из Конструкции 4.5, за исключением того, что в нем используется действительно случайная функция f вместо псевдослучайной функции Fk. То есть, $G\tilde{}en(1n)$ выбирает равномерную функцию $f \in Funcn$, и $M\tilde{}ac$ вычисляет тэг, так как Mac принимает то, что вместо Fk используется f. Из этого непосредственно следует, что

$$\Pr[\text{Mac-forge}_{A,\Pi}(n) = 1] \leq 2^{-n} \qquad (4.1)$$

потому что для любого сообщения $m \notin Q$, значение $t = f(m)$ равномерно распределено в $\{0,1\}n$ с точки зрения противника A.

Мы покажем, что существует пренебрежимо малая функция negl, такая что

$$\left|\Pr[\text{Mac-forge}_{\mathcal{A},\Pi}(n) = 1] - \Pr[\text{Mac-forge}_{\mathcal{A},\tilde{\Pi}}(n) = 1]\right| \leq \text{negl}(n); \qquad (4.2)$$

соединив это уравнение с Уравнением (4.1), получаем

$\Pr[\text{Mac-forge}_{A,\Pi}(n) = 1] \leq 2^{-n} + \text{negl}(n)$, что и требовалось доказать.

Чтобы доказать Уравнение (4.2), построим дистинктор полиномиального времени D, которому дан оракульный доступ к некоторой функции, и который имеет своей целью определить, является ли эта функция псевдослучайной (т.е. равной Fk для единообразного $k \in \{0,1\}n$) или случайной (т.е. равной f для единообразной $f \in Func_n$). Для этого D имитирует эксперимент аутентификации сообщения для A и проверяет, сможет ли A выдать действующий тэг для «нового» сообщения. В этом случае, D уга-

дывает, что его оракул является псевдослучайной функцией. В противном случае, D угадывает, что его оракул является случайной функцией. Более подробно:

Дистинктор D:

D получает вход 1^n и доступ к оракулу $O : \{0, 1\}^n \to \{0, 1\}^n$, и работает следующим образом:

1. Запустить $A(1^n)$. Каждый раз, когда A посылает запрос его оракулу КАСа для сообщения m (т.е. когда A запрашивает тэг для сообщения m), ответить на запрос следующим образом:

Послать запрос O с m и получить ответ t; вернуть A ответ t .

2. Когда A выдает (m, t) в конце своей работы, делать следующее:

(a) Послать запрос O с m и получить ответ t'.

(b) Если (1) t' = t и (2) A никогда не посылал своему оракулу КАСа запрос для m, то выдать 1; в противном случае выдать 0. Очевидно, что D работает в полиномиальном времени.

Заметим, что если оракул дистинктора Dявляется псевдослучайной функцией, то вид A , когда он запускается дистинктором D как субрутина, распределен идентично виду A в эксперименте $\text{Mac-forge}_{A,\Pi}(n)$. Кроме того, D выдает 1, именно когда $\text{Mac-forge}_{A,\Pi}(n) = 1$. Следовательно,

$$\Pr\left[D^{F_k(\cdot)}(1^n) = 1\right] = \Pr\left[\text{Mac-forge}_{\mathcal{A},\Pi}(n) = 1\right],$$

где $k \in \{0, 1\}^n$ выбирается равномерно. Если оракул D — это случайная функция, то вид A , когда он запускается дистинктором D как субрутина, распределен идентично виду в эксперименте $\text{Mac-forge}_{A,\Pi}(n)$ и снова D выдает 1 именно тогда, когда $\text{Mac-forge}_{A,\Pi}(n)=1$ A,Π'. Таким образом,

$$\Pr\left[D^{f(\cdot)}(1^n) = 1\right] = \Pr\left[\text{Mac-forge}_{\mathcal{A},\tilde{\Pi}}(n) = 1\right],$$

где $f \in \text{Func}_n$ выбирается равномерно.

Так как F — псевдослучайная функция, и D работает в полиномиальном времени, то существует пренебрежимо малая функция negl , такая что:

$$\left|\Pr\left[D^{F_k(\cdot)}(1^n) = 1\right] - \Pr\left[D^{f(\cdot)}(1^n) = 1\right]\right| \leq \text{negl}(n).$$

Из этого следует Уравнение (4.2), что и требовалось доказать.

---

[3] Если использовать псевдослучайную функцию, которая принимает на входе данные произвольной длины, Конструкция 4.5 может быть использована для построения КАС для сообщений произвольной длины. Таким же образом, псевдослучайная функция

4.3.2 Расширение области для кодов аутентификации сообщений

Конструкция 4.5 важна тем, что она показывает общую парадигму для построения безопасных кодов аутентификации сообщений из псевдослучайных функций. К сожалению, эта конструкция применима только к сообщениям фиксированной длины, которые к тому же достаточно короткие[3] .Эти ограничения неприемлемы в большинстве приложений. Мы покажем, как можно построить общий КАС, работающий для сообщений произвольной длины, из любого КАСа фиксированной длины для сообщений длины n. Изображенная конструкция не очень эффективна и поэтому не очень часто используется на практике. В действительности, известны намного более эффективные конструкции защищенных МАС, как было описано в Разделах 4.4 и 5.3.2. Мы общаемся к настоящей конструкции из-за ее простоты и универсальности.

Пусть $\Pi^r = (Mac^r, Vrfy^r)$ будет защищенным МАС фиксированной длины для сообщений длиной n. Перед представлением конструкции МАС для сообщений произвольной длины, основанных на Пr, мы исключим несколько простых идей и опишем несколько канонических атак, которые необходимо предотвратить. Ниже мы разберем сообщение m , чтобы аутентифицировать его как последовательность блоков m1, . . . , md; обратите внимание, что, так как нашей целью является обработка сообщения произвольной длины, d может варьироваться от сообщения к сообщению.

4.3.2.1 Стандартная первая идея заключается в том, чтобы просто аутентифицировать каждый блок по отдельности, то есть вычислить $ti := Mac^r(mi)$ для всех i и вывести (t1, . . . , td) в качестве тега. Это не позволит злоумышленнику отсылать ранее неаутентифицированный блок незаметно. Однако, это не спасет от атаки с изменением порядков блоков, при которой атакующий перетасовывает блоки в аутентифицированном сообщении. В частности, если (t1, t2) действительный тег на сообщении m1, m2 (при m1 $\neq$ m2), тогда (t2, t1) - действительный тег на (другом) сообщении m2, m1 (что-то, что не разрешено Определением 4.2).

4.3.2.2 Мы можем предотвратить предыдущую атаку путем аутентификации индекса блока вместе с каждым блоком. То есть мы теперь вычисляем $ti = Mac^r(i»mi)$ для всех i и выводим (t1, . . . , td) в качестве тега. (Обратите внимание, что длина блока |mi| должна измениться.) Это не предотвращает атаку с усечением, когда атакующий просто сбрасывает блоки с конца сообщения (и также сбрасывает соотвествующие блоки тегов).

4.3.2.3 Атаке с усечением можно помешать посредством дополнительной аутентификации длины сообщений вместе с каждым блоком. (Аутентификация длины сообщений в качестве отдельного блока не работает. Понимаете ли вы почему?) То

с большей областью даст защищенный МАС для более длинных сообщений. Однако, существующие прак тические псевдослучайные функции (то есть блочные шифры) принимают короткие данные фиксированной длины.

есть вычислите ti = Mac^r (A»i»mi) для всех i, где A обозначает длину сообщения в битах. (И снова же, длину блока |mi| необходимо будет уменьшить.) Данная схема уязвима для атаки с комбинированием блоков, когда злоумышленник объединяет блоки из различных сообщений. Например, если злоумышленник получит теги (t1, . . . , td) и (tr , . . . , tr) на сообщения m = m1, . . . , md and mr = mr , . . . , mr , соответственно, он может вывести действительный тег (t1, tr , t3, tr , . . .) на сообщениях m1, mr , m3, mr , ... Мы можем предотвратить последнюю атаку, также введя случайный «идентификатор сообщений» вместе с каждым блоком, что защитит блоки из разных сообщений от комбинирования. Это приводит нас к Конструкции 4.7.

КОНСТРУКЦИЯ 4.7

Пусть Пr = (Mac^r , $Vrfy^r$) будет MAC фиксированной длины для сообщения длиной n. Определяйте MAC следующим

Mac: при вводе ключ k ∈ {0, 1} и сообщение, m ∈ {0, 1} (ненулевой) длины $A < 2^{n/4}$, разберите m на d блоки m1, . . . , md, каждый длиной n/4. (Последний блок дополните нулями при необходимости.)

Выберите универсальный идентификатор r ∈ $\{0, 1\}^{n/4}$

Для i = 1, . . . , d, вычислите t ← Mac k(r"A"i"m), где i, A зашифрованы в виде строк длиной n/4.† Выведите тег t := (r, t1, . . . , td).

Vrfy: при вводе ключ k ∈ $\{0, 1\}^n$, сообщение m ∈ {0, 1}* и тег $A< 2^{n/4}$ и тег t = (r, t1, . . . , tdt), разберите m на d блоки m1, . . . , md, каждый длиной n/4. (Последний блок дополните нулями при необходимости.) Выведите 1, если и только если $d^r = d$ и Vrfyr (r" A"i" m , t) = 1 для 1 ≤ i ≤ d.

---

† Обратите внимание, что i и A могут быть зашифрованы с использованием n/4 бит, потому что i, A < 2n/4.

MAC для сообщений произвольной длины из любого MAC фиксированной длины.

(Технически, схема обрабатвает только лишь сообщения длиной менее $2^{n/4}$.) Асимптотически, так как это экспоненциальная граница, доверенные стороны не будут аутентифицировать сообщения такой длины, и любой полиномиально-временной злоумышленник не сможет отправить сообщение такой длины своему оракулу MAC. На практике, когда конкретное значение n фиксированное , необходимо убедиться, что эта граница приемлема.)

ТЕОРЕМА 4.8 *Если $П^r$ является защищенным* MAC *фиксированной длины для сообщений длиной n, тогда Конструкция 4.7 является защищенным* MAC *(для сообщений произвольной длины)*.

ДОКАЗАТЕЛЬСТВО Интуиция подсказывает, что пока $П^r$ под защитой, злоумышленник не может ввести новый блок с действительным тегом. Более того, дополнительная информация, включенная в каждый блок, предотвращает различные атаки (сброшенные блоки, переставленные блоки и т.д.), описанные

ранее. Мы докажем безопасность, фактически показав, что эти атаки являются единственными из возможных.

Пусть П будет MAC на основании Конструкции 4.7, и пусть А будет вероятностным полиноминально-временным злоумышленником. Мы покажем, что Pr[Mac-forge$_{АП}$(n) = 1] является пренебрежимо малым. Для начала мы представим определенное обозначение, которое будет использовано в доказательстве. Пусть Repeat обозначает событие, при котором один и тот же случайный идентификатор появляется в двух из тегов, возвращенных оракулом MAC в эксперименте Mac-forge$_{АП}$(n). Позволяя (m, t = (r, t1, . . .)) обозначать конечные выходные данные А, где m = m1, . . . имеет длину А, мы позволяем NewBlock быть событием, когда по крйней мере один из блоков r||A||i||mi никогда ранее не аутентифицировался посредством Macr в процессе ответа на А запросы Mac. (Обратие внимание, что по конструкции П легко сказать точно, какие блоки аутентифицировались посредством Macr , при вычислении Mack (m).) Неформально, NewBlock - это событие, когда А пытается вывести действительный тег на блоке, который никогда не был аутентифицирован соответствующим MAC Пr фиксированной длины.

Мы имеем

Pr[Mac-forge$_{АП}$(n) = 1] = Pr[Mac-forge$_{АП}$(n) = 1 ^ Repeat]
+ Pr[Mac-forge$_{АП}$(n) = 1 ^ eRwepBeloactk^ N]
+ Pr[Mac-forge$_{АП}$(n) = 1 ^ Repeat ^NewBlock] ≤ Pr[Repeat] (4.3)
+ Pr[Mac-forge$_{АП}$(n) = 1 ^ NewBlock]
+ Pr[Mac-forge$_{АП}$(n) = 1 ^ Repeat ^ NewBlock].

Мы покажем, что первые два члена Уравнения (4.3) пренебрежимо малы, а последний член - это 0. Из этого следует, что Pr[Mac-forge$_{АП}$(n) = 1] является пренебрежимо малым, как и планировалось .

УТВЕРЖДЕНИЕ 4.9 Pr[Repeat] является незначительным .

ДОКАЗАТЕЛЬСТВО Пусть q(n) будет количеством запросов оракула MAC, выполненных А. Чтобы ответить на запрос оракула i от А, оракул единообразно выбирает ri из набора размером $2^{n/4}$. Вероятность события Repeat равна вероятности того, что ri = rj для некоторых i *f*= j. Применяя «границу дней рождения» (Лемма А. (Lemma A.)15), мы имеем то, что Pr[Repeat] ≤ $\frac{q(n)^2}{2^{n/4}}$. . Поскольку А выполняет только полиномиально много запросов, это значение является пренебрежимо малым.

Далее мы рассмотрим последний член с правой стороны Уравнения (4.3). Мы утверждаем, что, если Mac-forge$_{АП}$(n) = 1, но Repeat не возникает, тогда это должно быть тот случай, когда возникает NewBlock. То есть Mac-forge$_{АП}$(n) = 1^ Repeat вызывает NewBlock, и поэтому

Pr[Mac-forge$_{АП}$(n) = 1 ^ Repeat ^ =Ne0wBlock].

Это, в какой-то степени, является сердцем доказательства.

И снова, пусть q = q(n) обозначает количество запросов оракула MAC, выполненных A, и пусть ri обозначает случайный идентификатор, используемый для ответа на запрос оракула i A. Если Repeat не возникает, тогда значения r1, . . . , rq будут разными. Пусть (m, t = (r, t1, . . .)) будут выходными данными A при m = m1, ... Если r $f \notin$ {r1, . . . , rq }, тогда явно явно NewBlock. Если нет, тогда r = rj для какого-то уникального j, а блоки r»A»1»m1, . . ., возможно, не будут аутентифицированы в процессе ответа какому-либо запросу Mac, отличному от такого запроса j. Пусть m(j) будет сообщением, использованным A для своего запроса оракула j, и пусть Aj будет его длиной. Для рассмотрения имеется два случая:

Случай1: A $f \neq$ Aj . Блоки, аутентифиицированные во время ответа на запрос Mac j, все имеют Aj $f \neq$ A во второй позиции. Поэтому r»A»1»m1,в частности, никогда не аутетифицировался во время ответа на запрос Mac j, и происходит NewBlock.

Case 2: A = Aj . Если Mac-forge$_{A\Pi}$(n)= 1, тогда мы должны иметь m f= m(j). Пусть 1 , ... Так как m и m(j) имеют идентичную длину, должен быть по крайней мере один индекс i, для которого mi $=f$ m(j). Блок r»A»i»mi, таким образом, никогда не был аутентифицирован в процессе ответа на запрос Mac j. (Потому что i включено в третью позицию блока, блок r»A»i»mi , возможно, мог бы быть аутентифицирован, если бы r»A»i»mi = rj «Aj «i»m(j), но это неверно, так как mi $f \neq$ m(j).)

Чтобы закончить доказательство теоремы, мы ограничим второй член правой стороны Уравнения (4.2):

УТВЕРЖДЕНИЕ 4.10 Pr[Mac-forge$_{A\Pi}$(n) = 1 ^ NewBlock] является пренебрежимо малым.

Утверждение основывается на безопасности Π^r. Мы сконструировали ppt злоумышленника A^r, который атакует MAC Π^r фиксированной длины и добивается успеха в выведении действительной подделки прежде неаутентифицированного сообщения с вероятностью

Pr[Mac-forge$_{A^r\Pi^r}$(n) = 1] $\geq$ Pr[Mac-forge$_{A\Pi}$(n) = 1 ^ NewBlock]. (4.4)

Безопасность Π^r означает, что левая стороны пренебрежимо мала, доказывая утверждение. Конструкция A^r вполне очевидна, поэтому мы описываем ее вкратце. A^r запускает A в качестве подпрограммы и отвечает на запрос от A касательно тега на m, подбирая r $\leftarrow$ {0, 1}$^{n/4}$, разбирая m соответствующим образом, и выполняя необходимые запросы своему собственному оракулу MAC Macr (•). Когда A выводит (m, t =(r, t1, . . .)), тогда A^r проверяет, случиться ли NewBlock (это легко сделать, поскольку A^r может отслеживать все запросы, которые он делает своему собственному оракулу). Если так, тогда A^r находит первый блок r»A»i»mi, который никогда ранее не аутентифицировался посредством Macr, и выводит (r»A»i»mi, ti). (Если нет, A^r не выводит ничего.)

Видимость A , когда он запущен в качестве подпрограммы посредством A^r, распространяется идентично видимости A в эксперименте $Mac\text{-}forge_{A\Pi}(n)$, и поэтому вероятности событий Mac-forge A(n) = 1, а NewBlock не меняется. Если случается NewBlock , тогда A^r выводит блок r||A||i||mi, который никогда ранее не был аутентифицирован собственным оракулом MAC; если $Mac\text{-}forge_{A\Pi}(n)$ = 1, тогда тег на каждом блоке будет действительным (относительно Π^r), и поэтому, в частности, это истинно для блока, выведеного посредством A^r. Это означает, что не имеет значение, верно ли, что $Mac\text{-}forge_{A\Pi}(n) = 1$, случается ли NewBlock, мы имеем $Mac\text{-}forge_{A\Pi}(n) = 1$, доказывая Уравнение (4.4).

4.4 CBC-MAC

Теоремы 4.6 и 4.8 показывают, что можно создать защищенный код аутентификации сообщений для сообщений произвольной длины из псевдослучайной функции, прини мающей данные фиксированной длины n. Это, в принципе, демонстрирует, что безопасные КАС могут быть созданы из блочных шифров. К сожалению, полученная конструкция чрезвычайно неэффективна: чтобы вычислить тег на сообщении длиной dn, блочный шифр вычисляется 4d раз; тег - более 4dn бит в длину. К счастью, доступны намного более эффективные конструкции. Мы изучаем одну такую конструкцию здесь, которая основывается исключительно на блочных шифрах, а еще одну - в Разделе 5.3.2, которая использует дополнительный криптографический примитив.

4.4.1 Базовая конструкция

CBC-MAC является стандартизированным кодом аутентификации сообщений, широко используемым на практике. Базовая версия CBC-MAC, защищенного при аутентификации сообщений любой фиксированной длины, приведена в качестве Конструкции 4.11. (См. также Рисунок 4.1.) Предупреждаем, что базовая схема незащищена в стандартных случаях, когда сообщения различной длины могут быть аутентифицированы; смотрите дальнейшее описание ниже.

КОНСТРУКЦИЯ 4.11

Пусть F будет псевдослучайной функцией, зафиксируйте функцию длины A > 0. Базовая конструкция CBC-MAC такова:

• Mac: при вводе ключ $k \in \{0, 1\}$ и сообщение m длиной A(n)•n, сделайте следующее (мы установим A = A(n) далее):

1.Разберите m как m = m1, . . . , mÆ, где каждое mi имеет длину n.

2. Задайте t0 := 0n. Затем для i = 1 A: Задайте ti := Fk(ti−1 ⊕ mi).

Выведите tÆ в качестве тега.

• Vrfy: при вводе ключ $k \in \{0, 1\}$, сообщение m и тег t, сделайте: Если m не имеет длину A(n) • n, тогда выведите 0. Иначе, выведите 1, если и только если t =? Mac (m).

Базовый CBC-MAC (для сообщений фиксированной длины).

ТЕОРЕМА 4.12 *Пусть A будет полиномиальным. Если F является псевдослучайной функцией, Конструкция 4.11 является защищенным MAC для сообщений длинной A(n) • n.*

Доказательство Теоремы 4.12 довольно сложное. В следующем разделе мы покажем более общий результат, из которого и следует вышеупомянутая теорема.

Хотя Конструкция 4.11 может быть явным образом расширена для обработки сообщений, длина которых произвольно кратная n, конструкция только тогда защищена, когда длина сообщений для аутентификации фиксированная и заранее согласованная между отправителем и получателем. (См. Упражнение 4.13.)

Преимущество данной конструкции по сравнению с Конструкцией 4.5, которая также дает MAC фиксированной длины, в том, что настоящая конструкция может аутентифицировать более длинные сообщения. В сравнении с Конструкцией 4.7, CBC-MAC намного более эффективная, требующая только вычисления блочного шифра d для сообщений длиной dn и с тегом только лишь длины n.

CBC-MAC против шифрования в режиме CBC. CBC-MAC похож на режим использования CBC. Однако, существуют некоторые важные различия:

1. Шифрование в режиме CBC использует случайный IV, и это является ключевой особенностью для безопасности. Для сравнения, CBC-MAC не использует IV (альтернативно, он может рассматриваться как использующий фиксированное значение IV = 0n), и это также ключевая особенность для безопасности. В частности, CBC-MAC, использующий случайный IV, не является защищенным.

2. При шифровании в режиме CBC все промежуточные значения ti (под названием ci в случае шифрования в режиме CBC) выводятся шифровальным алгоритмом как часть шифротекста, тогда как при использовании CBC-MAC выводится только последний блок в качестве тега. Если CBC-MAC модифицируется так, чтобы выводить все {ti}, полученные в процессе вычисления, тогда он перестает быть защищенным.

В упражнении 4.14 вам необходимо удостовериться, что модификации CBC-MAC, описанные выше, не являются защищенными. Данные примеры демонстрируют тот факт, что на вид безобидные модификации криптографических конструкций могут сделать их незащищенными. Всегда внедряйте криптографические конструкции строго по инструкции и не модифицируйте их (если только такие модификации сами по себе не являются защищенными). Более того, важно понимать конструкцию, которую вы используете. Во многих случаях криптографическая библиотека предоставляет программисту «функцию CBC», но не различает использование данной функции для шифрования или аутентификации сообщений.

ЗащищенныйCBC-MAC для сообщений произвольной длины . Мы кратко описали два способа, которыми можно модифицировать Конструкцию 4.11,

возможно, даже безопасно, чтобы обрабатывать сообщения произвольной длины. (Здесь для простоты мы предположим, что сообщения для аутентификации имеют длину кратную n, и что Vrfy отклоняет

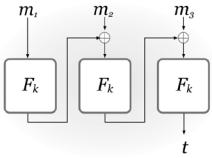

РИСУНОК 4.1: Базовый CBC-MAC (для сообщений фиксированной длины).

любое сообщение, длина которого не кратная n. В следующем разделе мы рассмотрим более общий случай, при котором сообщения могут иметь произвольную длину.)

1. Дополните начало сообщения m его длиной |m| (зашифровано как n-битная строка), затем вычислите базовый CBC-MAC по результату; см. Рисунок 4.2. Безопасность такого варианта исходит из результатов, доказанных в следующем разделе.

Обратите внимание, что добавление |m| в конец сообщения, а затем вычисление базового CBC-MAC небезопасно.

2. Измените схему так, чтобы генерирование ключа подобрало два независимых универсальных ключа k1 ∈ {0, 1}n and k2 ∈ {0, 1}n. Затем для аутентифиуации сообщения m, сначала вычислите базовый CBC-MAC m, используя k1 , и пусть t будет результатом; выведите тег tˆ := Fk2 (t).

Второй вариант имеет преимущество в том, что нет нужды знать длину сообщение заранее (то есть в начале вычисления тега). Однако, есть и недостаток в использовании ключей для F. Обратите внимание, что за счет двух дополнительных применений псевдослучайной функции, можно сохранить единственный ключ k и затем получить ключи k1 := Fk (1) и k2 := Fk (2) в начале вычислений. Несмотря на это, в практике операция инициализации ключа для блочного шифра считается относительно дорогостоящей. Таким образом, необходимость в двух ключах, даже если они извлекаются динамически, является менее целесообразной.

4.4.2 *Доказательство беопасности

В этом разделе мы докажем безопасность различных вариантов CBC-MAC. Начнем с обобщения результатов, а затем выложим детали доказательства. Перед началом мы отметим, что доказательство в данном разделе достаточно сложное и предназначено для продвинутых читателей.

По всему этому разделу, зафиксируйте ключевую функцию F , которая, учитывая параметр защиты n, преобразовывает n-битные ключи и n-битные входные данные в n-битные выходные данные. Мы определим ключевую

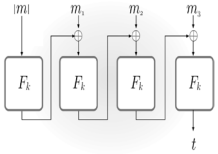

РИСУНОК 4.2: Вариант CBC-MAC, защищенного для аутентификации сообщений проивольной длины.

функцию CBC , которая, учитывая параметр защиты n, преобразовывает n-битные ключи и входные данные в ({0, 1}n)* (то есть строки, длина которых кратная n) в n-битные выходные данные. Данная функция определяется как

$$\mathrm{CBC}\ (x\ ,\ldots,x) \stackrel{\mathrm{def}}{=\!=} F\ (F\ (\bullet\bullet\bullet F\ (F\ (x\ ) \oplus x\ ) \oplus \bullet\bullet\bullet) \oplus x\ ),$$

где |x1| = • • • = |xA | = |k| = n. (Мы оставляем CBCk неопределенной на пустую строку.) Обратите внимание, что CBC точь в точь базовый CBC-MAC, хотя здесь мы рассматриваем входные данные различной длины.

Набор строк P ∈ ({0, 1}n)* является бе зпрефиксным, если он не содержит пустых строк, и ни одна строка X ∈ P не является префиксом какой-либо другой строки Xr ∈ P . Покажем:

ТЕОРЕМА 4.13 Если F является псевдослучайной функцией, тогда и CBC является псевдослучайной функцией, поскольку набор входных данных, который она опрашивает, является беспрефиксным. Строго говоря, для всех вероятностных полиномиально-временных идентификаторов D , которые опрашивают своего оракула касательно беспрефиксного набора входных данных, существует пренебрежимо малая функция negl такая, что

$$\left| \Pr[D^{\mathsf{CBC}_k(\cdot)}(1^n) = 1] - \Pr[D^{f(\cdot)}(1^n) = 1] \right| \leq \mathsf{negl}(n),$$

где k выбирается единообразно из $\{0, 1\}^n$, а f выбирается единообразно из набора функций, преобразующих ({0, 1}n)* в $\{0, 1\}^n$ (то есть значение f при каждом входном элементе является универсальным и независимым от значений f при всех остальных входных данных).

Таким образом, мы можем преобразовать псевдослучайную функцию F для входных данных фиксированной длины в псевдослучайную функцию CBC для входных данных произвольной длины (подчиняющихся условию, по которому входные данные были запрошены)! Чтобы использовать это для аутентификации сообщений, мы адаптируем идею Конструкции 4.5 следующим образом: чтобы аутентифицировать сообщение m, для начала применим некоторую функцию шифрования encode, чтобы получить (непустую) строку encode(m) ∈ ({0, 1}n)*; затем выведем тег CBCk (encode(m)). Здесь, чтобы обеспечить защищенность (сравните доказательством Теоремы 4.6), шифрование должно быть без префикс-

ным, а именно, иметь такое свойство, чтобы для любого нового (законного) сообщения m1, m2, строка encode(m1) не являлась префиксом строки encode(m2). Это подразумевает то, что для любого набора (законных) сообщений {m1, . . .}, набор зашифрованных сообщений {encode(m1), . . .} будет беспрефиксным.

Давайте теперь изучим два конкретных применения этой идеи:

• Зафиксируйте A, и пусть набор законных сообщений будет $\{0,1\}^{A(n)\bullet n}$. Затем мы можем взять тривиальное шифрование encode(m) = m, которое является беспрефиксным, поскольку строка не может быть префиксом другой строки той же длины. Это точь в точь базовый CBC-MAC, и как мы уже говорили выше, это подразумевает, что базовый CBC-MAC является защищенным для сообщений любой фиксированной длины (сравните с Теоремой 4.12).

• Один из способов обработки (непустых) сообщений произвольной длины (технически, сообщений длиной меньше чем 2n) - это зашифровать строку $m \in \{0,1\}^*$ посредством добавления в начало строки ее длины |m| (зашифрованной n-битной строкой), а затем добавления в конец строки стольких нулей, сколько будет нужно, чтобы сделать длину итоговой строки кратной n. (Это очень важно, что и показано на Рисунке 4.2.) такое шифрование является беспрефиксным, и мы, таким образом, получаем защищенный MAC для сообщений произвольной длины.

Оставшаяся часть раздела посвящена доказательству Теоремы 4.13. Доказывая теорему, мы анализируем CBC , когда она «ключевая» при случайной функции g , а не случайном ключе k для какой-то присутствующей псевдослучайной функции F . То есть мы рассматриваем ключевую функцию CBCg , определенную как

$$\mathrm{CBC}(x_1, \ldots, x_\ell) \stackrel{\mathrm{def}}{=} g(g(\bullet\bullet\bullet g(g(x_1) \oplus x_2) \oplus \bullet\bullet\bullet) \oplus x_\ell)$$

где, учитывая параметр защиты n, функция g преобразовывает n-битные входные данные bn-битные выходные данные, и $|x_1| = \bullet\bullet\bullet = |x_A| = n$. Обратите внимание, что CBCg , как было здесь определено, не является эффективной (поскольку представление g требует экспоненциала пространства в n); тем не менее, она все еще хорошо определенная, ключевая функция.

Мы покажем, что если g выбирается единообразно из Funcn, то CBCg является неотличимой от случайной функции, преобразующей $(\{0,1\}^n)^*$ в n-битные строки при условии, что опрашивается беспрефиксный набор входных данных. Более точно:

УТВЕРЖДЕНИЕ 4.14 Зафиксируем $n \geq 1$. Для всех идентификаторов D , которые опрашивают своих оракулов касательно беспрефиксных наборов q входных данных, в которых самый длинный такой входной элемент содержит A блоков, справедливо, что:

$$\left| \Pr[D^{\mathsf{CBC}_g(\cdot)}(1^n) = 1] - \Pr[D^{f(\cdot)}(1^n) = 1] \right| \leq \frac{q^2 \ell^2}{2^n},$$

где g выбирается единообразно из Funcn, а f выбирается единообразно из набора функций, преобразовывающих ({0, 1}n)* в {0, 1}^{n}.

(Утверждение ничем не обусловлено и не накладывает никаких ограничений на время работы D. Таким образом, мы можем заставить D быть детерминированной.) Вышесказанное подразумевает Теорему 4.13, используя стандартные методы, которые мы уже проходили. В частности, для любых D, работающих в полиномиальном времени, мы должны иметь q(n), A(n) = poly(n), и поэтому q(n)^{2}A(n)^{2} • $^{2-n}$ является незначительным .

ДОКАЗАТЕЛЬСТВО (Утверждения 4.14) Зафиксируем n ≥ 1. Доказательство происходит в два этапа: Для начала определим понятие гладкость и докажем, что CBC является гладкой; затем мы покажем, что гладкость подразумевает утверждение.

Пусть P = {X1, . . . , Xq } будет беспрефиксным набором q входных данных, каждый Xi в ({0, 1}^{n})*, а самая длинная строка вP содержит A блоков (то есть каждый Xi ∈ P содержит максимум A блоков длиной n). Обратите внимание, что для t1, . . . , tq ∈ {0, 1}n ісправедливо, что Pr [∀i i : f (Xi) = ti] = 2^{-nq}, когда вероятность выше единообразного выбора функции f из набора функций, преобразующих ({0, 1}n)* в {0, 1}n. Скажем, что CBC является (q, A, δ)-гладкой, если для каждого беспрефиксного набора P = {X1, . . . , Xq } , как сказано выше, и каждого t1, . . . , tq ∈ {0, 1}n, справедливо, что $\Pr[\forall i : \mathsf{CBC}_g(X_i) = t_i] \geq (1-\delta) \cdot 2^{-nq}$, когда вероятность выше единообразного выбора g ∈ Funcn .

Другими словами, CBC является гладкой, если для каждого фиксированного набора входных/выходных пар {(Xi, ti)}, где {Xi} формирует беспрефиксный набор, вероятность того, что CBCg (Xi) = ti для всех i - δ-близка к вероятности того, что f (Xi) = ti для всех i (где g является случайной функцией от {0, 1}n в {0, 1}^{n}, а f является случайной функцией из ({0, 1}^{n})* до {0, 1}^{n}).

УТВЕРЖДЕНИЕ 4.15 CBCg является (q, A, δ)-гладкой, потому что δ = q^{2}A^{2} • $^{2-n}$.

ДОКАКАЗАТЕЛЬСТВО Для любых X ∈ ({0, 1}n)*, при X = x1, . . . и xi ∈ {0, 1}^{n}, пусть Cg (X) обозначает набор входных данных, по которым g высчитывается во время расчета CBCg (X); то есть, еслиX ∈ ({0, 1}^{n})m , тогда

Cg (X) $\stackrel{\text{def}}{=}$ (x1, CBCg (x1) ⊕ x2, . . . , CBCg (x1, . . . , xm−1) ⊕ xm) .

Для X ∈ ({0, 1}^{n})^{m} и X^{r} ∈ ({0, 1}^{n})^{m}, при Cg (X) = (I1, . . . , Im) и

Cg (X^{r}) = (I^{r}, . . . , I^{r}), скажем, существует нетривиальная коллизия в X , если Ii = Ij для 1 mt некоторых i *f*= j, а также скажем, что существует нетривиальная коллизия между X и Xr , если Ii = I^{r} , но (x1, . . . , xi) *f*= (x^{r} , . . . , x^{r}) (в этом последнем случае i может равняться j). Скажем, j 1 j что существует нетривиальная коллизия в P , если существует нетривиальная коллизия в некоторых X∈ P или между некоторыми парами строк X, Xr ∈ P . Пусть Coll будет фактом, что

существует нетривиальная коллизия в P .

Докажем утверждение в два этапа. Во-первых, мы покажем, что обусловленная тем, что в P нет тривиальной коллизии, вероятность того, что CBCg (Xi) = ti для всех i ровно 2−nq . Затем мы покажем, что вероятность того, что существует нетривиальная коллизия в P менее чем $\delta = q^2A^2 \cdot 2^{-n}$.

Рассмотрим, выбирая универсальную g , посредством выбора, одного за другим, универсальных значений для выходных параметров g касательно разных входных параметров. Определение того, существует ли нетривиальная коллизия между вумя строками X, Xr ∈ P , может быть выполнено для начала путем выбора значений для g(I1) и $g(I^r)$ (если $I^r = I1$, эти значения идентичны), а затем выбора 11 значений для g(I2) и g(Ir) (обратите внимание, что $I2 = g(I1) \oplus x2$ и $I^r = g(Ir) \oplus x^r$ определяются после того, как g(I1), $g(I^r)$ были зафиксированы), и продолжая в том же духе, пока мы не выберем значения для g(Im−1) и $g(I^r)$. В частности, значения g(Im), $g(I^r)$ не обязательно выбирать для того, чтобы определить, существует ли нетривиальная коллизия между X и Xr. Продолжая данную цепочку рассуждений, можно определить, случается ли Coll при выборе значений g по всем, кроме последних, записей каждого из Cg (X1), . . . , Cg (Xq).

Предположим, что Coll не случилась после фиксирования значений g касательно разных входных данных, как описано выше. Рассмотрим последние записи в каждом из Cg (X1), . . . , Cg (Xq). Эти записи все разные (это и так было понятно из факта, что Coll не случилась), и мы заявляем, что значение g по каждой из тез точек все еще не было зафиксировано. В действительности, единственным способом, которым значение g могло бы быть уже зафиксировано по любой из тех точек, стало бы условие, если последняя запись Im некоторых Cg (X) равна непоследней записи Ij некоторых Cg (Xr). Но поскольку Coll не случилась, это может только случиться, если X *f*= Xr и (xr , . . . , xr) = (x1, . . . , xm). Но тогда X стала бы 1 префиксом Xr, нарушающим предположение, что P является беспрефиксной.

Так как g является случайной функцией, вышесказанное означает, что CBCg (X1), . . . , CBCg (Xq) являются универсальными и независимыми от других, равно как и все другие значения g , которые уже были зафиксированы. (Это потому что CBCg (Xi) является значением g , после вычисления при последней записи Cg (Xi), входной значение отличается от всех других входных данных, при которых g уже была зафиксирована.) Таким образом, для любых t1, . . . , tq ∈ {0, 1}n мы имеем:

$$\Pr\left[\forall i : \mathsf{CBC}_g(X_i) = t_i \mid \overline{\mathsf{Coll}}\right] = 2^{-nq}. \qquad (4.5)$$

Далее мы покажем, что cCcoull случается с высокой вероятностью верхней границы Pr[Coll]. Для разных Xi, Xj ∈ P , пусть $\mathrm{Coll}_{i,j}$ обозначает факт, что существует нетривиальная коллизия в X или X^r, или нетривиальная коллизия между X и X^r. Мы имеем $\mathrm{Coll} = V_{i,j}\mathrm{Coll}_{i,j}$, и поэтому граница объединения дает

$$\Pr[\mathsf{Coll}] \le \sum_{i,j:\ i<j} \Pr[\mathsf{Coll}_{i,j}] = \binom{q}{2} \cdot \Pr[\mathsf{Coll}_{i,j}] \le \frac{q^2}{2} \cdot \Pr[\mathsf{Coll}_{i,j}]. \quad (4.6)$$

Фиксируя разные X = Xi и Xr = Xj в P , мы сейчас ограничиваем $\mathrm{Coll}_{i,j}$. Как будет видно из анализа, вероятность максимизируется, когда X и X^r вместе как можно дольше, и, таким образом, мы предполагаем, что каждая из них A блоков в длину. Пусть X = (x1, . . . , xA) и $X^r = (x^r, \ldots, x^r)$, и пусть t будет наибольшим целым числом так, 1 A чтобы (x1, . . . , xt) = $(x^r, \ldots, x^r)$. (Обратите внимание, что t < A или X = Xr.) Мы предполагаем, что 1 t t > 0, но анализ, приведенный ниже, может быть с легкостью изменен, давая при этом такой же результат, если t = 0. Мы продолжаем, пусть I1, I2, . . . (соотвественно, I^r , I^r , . . .) обозначают входные данные 1 2 в g во время процесса вычисления CBCg (X) (соответственно, CBCg (X^r)); обратите внимание, что (I^r , . . . , I^r) = (I1, . . . , It). Рассмотрим выбор g посредством выбора универсальных значений для 1t выходных данных g, одного за одним. Мы сделаем это за 2A − 2 шагов следующим образом:

Шаги 1 через t − 1 (если t > 1): На каждом шаге i, выбираем универсальное значение для g(Ii), таким образом, определяя I_{i+1} и I^r_{i+1} (которые равны).

Шаг t: Выберем универсальное значение для g(It), таким образом, определяя I_{t+1} и I^r_{t+1}.

Шаги t + 1 до A − 1 (если t < A − 1): Выбираем, в свою очередь, универсальные значения для каждого из g(It+1), g(It+2), . . . , g(IA−1), таким образом, определяя I_{t+2}, I_{t+3}, . . . , IA .

Шаги A до 2A − 2 (если t < A − 1): Выбираем, в свою очередь, универсальные значения для каждого из $g(I'_{t+1})$, $g(I'_{t+2})$, . . . , $g(I'_{\ell-1})$, таким образом, определяя I'_{t+2}, I'_{t+3}, . . . , I'_{ℓ}..

Пусть Coll(k) будет фактом, что нетривиальная коллизия случается на шаге k. Тогда

$$\Pr[\mathsf{Coll}_{i,j}] = \Pr\left[\bigvee_k \mathsf{Coll}(k)\right] \le \Pr[\mathsf{Coll}(1)] + \sum_{k=2}^{2\ell-2} \Pr[\mathsf{Coll}(k) \mid \overline{\mathsf{Coll}(k-1)}], \quad (4.7)$$

используя Предположение A.9. Для k < t, мы заявляем, $\Pr[\mathsf{Coll}(k) \mid \overline{\mathsf{Coll}(k-1)}] = k/2^n$: в действительности, если не случилось никакой нетривиальной коллизии на шаге k − 1, значение g(Ik) выбирается единообразно на шаге k; нетривиальная коллизия случается, только если слечается так, что $I_{k+1} = g(Ik) \oplus x_{k+1}$ равняется оному из $\{I_1, \ldots, I_k\}$ (которые все разные, поскольку Coll(k − 1) не случилась). Согласно такому же рассуждению, мы имеем $\Pr[\mathsf{Coll}(t) \mid \overline{\mathsf{Coll}(t-1)}] \le 2t/2^n$ (здесь два значения I_{t+1}, I^r_{t+1} для рассмотрения; обратите внимание, что они не могут равняться друг другу). Наконец, аргументируя, как и ранее, для k > t мы имеем

$\Pr[\mathsf{Coll}(k) \mid \overline{\mathsf{Coll}(k-1)}] = (k+1)/2^n$. Уравнение (4.7), мы, таким образом, имеем

$$\Pr[\mathsf{Coll}_{i,j}] \le 2^{-n} \cdot \left(\sum_{k=1}^{t-1} k + 2t + \sum_{k=t+1}^{2\ell-2} (k+1) \right)$$
$$= 2^{-n} \cdot \sum_{k=2}^{2\ell-1} k \;=\; 2^{-n} \cdot (2\ell+1) \cdot (\ell-1) \;<\; 2\ell^2 \cdot 2^{-n}.$$

Из Уравнения (4.6) мы получаем $\Pr[\mathrm{Coll}] < q^2A^2 \bullet 2^{-n} = \delta$. Наконец, используя Уравнени (4.5) мы видим, что

$$\Pr\left[\forall i : \mathsf{CBC}_g(X_i) = t_i\right] \ge \Pr\left[\forall i : \mathsf{CBC}_g(X_i) = t_i \mid \overline{\mathsf{Coll}}\right] \cdot \Pr[\overline{\mathsf{Coll}}]$$
$$= 2^{-nq} \cdot \Pr[\overline{\mathsf{Coll}}] \;\ge (1-\delta) \cdot 2^{-nq},$$

как и утверждалось.

Мы теперь видим, что гладкость подразумевает теорему. Предположим, без потери обобщенности, что D всегда совершает q (разных) запросов, каждый из которых содержит максимум A блоков. D может выбиратьсвои запросы адаптивно (то есть в зависимости от ответов на рпедыдущие запросы), но набор запросов D должен быть беспрефиксным.

Дt ля разных $X_1, \ldots, X_q \in (\{0,1\}^n)^*$ и произвольных $t_1, \ldots, t_q \in \{0,1\}^n$, определим, что $\alpha(X_1, \ldots, X_q ; t_1, \ldots, t_q)$ равно 1, если и только если D выводит 1, при запросах $X_1, \ldots, X_q$ и полчении ответов $t_1, \ldots, t_q$. (Если, скажем, D не делает запрос X_1 как свой первый запрос, тогда $\alpha(X_1, \ldots ; \ldots) = 0$.) Допуская, что $X^{\cdot} = (X_1, \ldots, X_q)$ и $^{\cdot}t = (t_1, \ldots, t_q)$, мы, таким образом, имеем

$$\Pr[D^{\mathsf{CBC}_g(\cdot)}(1^n) = 1] = \sum_{\vec{X} \text{ prefix-free; } \vec{t}} \alpha(\vec{X}, \vec{t}) \cdot \Pr[\forall i : \mathsf{CBC}_g(X_i) = t_i]$$
$$\ge \sum_{\vec{X} \text{ prefix-free; } \vec{t}} \alpha(\vec{X}, \vec{t}) \cdot (1-\delta) \cdot \Pr[\forall i : f(X_i) = t_i]$$
$$= (1-\delta) \cdot \Pr[D^{f(\cdot)}(1^n) = 1],$$

где, выше, g выбирается единообразно из Funcn, а f выбирается единообразно из набора функций, преобразовывающих $(\{0,1\}^n)^*$ в $\{0,1\}^n$. Из этого следует

$$\Pr[D^{f(\cdot)}(1^n) = 1] - \Pr[D^{\mathsf{CBC}_g(\cdot)}(1^n) = 1] \le \delta \cdot \Pr[D^{f(\cdot)}(1^n) = 1] \le \delta.$$

Симметричный аргумент для того, когда D выводит 0, завершает доказательство.

4.5 Аутентифицированное шифрование

В Главе 3 мы изучали, как можно добиться стойкости в условиях шифрования с закрытым ключом. В этой главе мы показали, как обеспечить *целостность*, используя коды аутентификации сообщений. Кто-то естественно может захотеть достичь обеих целей одновременно, и это проблема, которой мы сейчас и займемся.

Лучше всего всегда *обеспечивать стойкость и целостность по умолчанию* в условиях закрытого ключа. Действительно,во многих приложениях, где необходима стойкость, оказывается, что целостность также важна. Более того, недостаток целостности может иногда привести к нарушению стойкости.

4.5.1 Определения

Мы начинаем, как обычно, с точного определения того, чего же мы желаем добиться. На абстрактном уровне нашей целью является реализация «безупречно защищенного» канала связи, которые обеспечивает как безопасность, так и целостность. Прес ледование определения такого рода находится за пределами данной книги. Вместо этого, мы предусматриваем более простой набор определений, которые обеспечивают стойкость и целостность по отдельности. Этих определений и нашего последующего анализа достаточно для понимания неизбежных ключевых проблем. (Мы предостерегаем читателя, однако, что, в отличие от шифрования и кодов аутентификации сообщений, область не установила еще стандартной терминологии и определений для аутентифицированного шифрования.)

Пусть Π = (Gen, Enc, Dec) будет схемой шифрования с закрытым ключом. Как уже было сказано, мы определяем безопасность отдельными определениями стойкости и целостности. Понятие стойкости, которое мы рассматриваем, идентичное тому, что мы видели и ранее: нам нужно было, чтобы Π был защищенным прости атак на основе подобранного шифротекста, то есть, чтобы он был CCA-защищенным. (Отправляйтесь в Раздел 3.7 для описания и определения защиты от атак на основе подобранного шифротекста.) Нас беспокоят атаки на основе подобранного шифротекста, потому что мы явно рассматриваем активного злоумышленника, который изменяет данные, направляемые от одной доверенной стороны другой доверенной стороне. Наше понятие целостности будет принципиально из конкретной невозможности подделки в условиях атаки на основании адаптивного подобранного открытого текста. Поскольку Π не удовлетворяет синтаксис кода аутентификации сообщений, однако, мы вводим определение специально для этого случая. Рассмотрим следующий эксперимент, определенный для схемы шифрования с закрытым ключом Π = (Gen, Enc, Dec), злоумышленника A и значения n параметра защиты:

Эксперимент устойчивого к подделке шифрования Enc-Forge$_{A\Pi}$(n) :

4.5.1.1 Запустим Gen(1^n) *, чтобы получить ключ k.*

4.5.1.2 Злоумышленник A получил входной элемент 1^n и получил доступ к оракулу шифрования Enck(•). Злоумышленник выводит шифротекст c.

4.5.1.3 Пусть m := Deck (c), и пусть Q обозначает набор все запросов, которые A делал своему оракулу шифрования. Результат эксперимента - 1, если и только если (1) m f=≠ и (2) m f∈ Q.

ОПРЕДЕЛЕНИЕ 4.16 *Схема шифрования с закрытым ключом П не поддается подделке, если для всех вероятностных плиномиально-временных злоумышленников А существует пренебрежимо малая функцияnegl такая, что:*

$Enc\text{-}Forge_{A\Pi}(n) = 1] \leq negl(n)$.

Параллельно вспоминая описание проверки запросов согласно Определению 4.2, здесь можно также рассмотреть более сильное определение, в котором А дополнительно имеет доступ к оракулу шифрования. Можно удостовериться, что защищенная конструкция, которую мы представим ниже, также удовлетворяет тому более сильному определению. Теперь мы определим (защищенную) схему аутентифицированного шифрования.

ОПРЕДЕЛЕНИЕ 4.17 Схема шифрования с закрытым ключом является схемой аутентифицированного шифрования, если она защищена с точки зрения атаки на основе подобранного шифрованного текста и не поддающаяся подделке.

4.5.2 Обобщенные конструкции

Хочется думать, что из любой оптимальной комбинации схемы защищенного шифрования и кода аутентификации сообщений должна получиться схема аутентифицированного шифрования. В этом разделе мы покажем, что это не тот случай. Это говорит о том, что даже защищенные криптографические инструменты могут быть объединены таким образом, чтобы получилась небезопасная конструкция, что лишний раз подтверждает важность определений и доказательств безопасности. С другой стороны мы покажем, как шифрование и аутентификация сообщений можно успешно совместить, чтобы достичь стойкости и целостности.

Без исключения, пусть ПЕ = (Enc, Dec) будет схемой шифрования, защищенной от атак на основе подобранного открытого текста, и пусть ПМ = (Mac, Vrfy) отображает код аутентификации сообщений, в котором генерирование ключей в обеих схемах использует просто подбор универсального n-битного ключа. Существует три стандартных подхода для объединения шифрования и аутентификации сообщений с использованием независимых ключей [4] k_E иand k_M для Π_E и Π_M, соотвественно:

4.5.2.1 Шифрование и аутентификация: В этом методе шифрование и аутентификация вычисляются параллельно и независимо. То есть имея сообщение из открытого текста m, отправитель передает шифротекст (c, t), где:

$c \leftarrow Enc_{kE}(m)$ и $t \leftarrow Mac_{kM}(m)$.

Получатель дешифрует c , чтобы извлечь m; предположив, что ошибок не случилось, он затем проверяет тег t. Если VrfykM (m, t) = 1, получатель выводит m; иначе, выв одится ошибка.

4независимые криптографические ключи должны всегда использоваться при комбинировании различных схем . Мы вернемся к этому вопросу в конце раздела.

4.5.2.2 Аутентификация, затем шифрование: Вот впервые был вычислен тег MAC t , и затем сообщение и тег защифровуются вместе. То есть дано сообщение m, отправитель передает шифротекст c , вычисленный как:

$t \leftarrow Mac_{kM}(m)$ и $c \leftarrow Enc_{kE}(m||t)$.

Получатель расшифровывает c, чтобы получить m»t; предположив, что ошибок не случилось, он затем проверяет тег t. Как и ранее, если VrfykM (m, t) = 1, получатель выводит сообщение m; иначе, выводится ошибка.

4.5.2.3 Шифрование, затем аутентификация: В этом случае сообщение m сначала шифруется, и затем тег MAC высчитывается по результату. То есть шифротекст идет в паре (c, t), где: $c \leftarrow Enc_{kE}(m)$ и $t \leftarrow Mac_{kM}(c)$.

(См. также Конструкцию 4.18.) Если VrfykM (c, t) = 1, тогда получатель дешифрует c и выводит результат; иначе, выводится ошибка.

Мы анализируем каждый из вышеописанных подходов , когда они реализованы посредством «типичных» компонентов безопасности, то есть *злоумышленник*, схема CPA-защищенного шифрования и *произвольный* (строго) защищенный MAC. Мы хотим подход, который обеспечит как стойкость, так и целостность, при использовании любых (защищенных) компонентов, и мы, таким образом, отклоним как «небезопасный» любой подход, для которого существует хотя бы одна схема защищенного шифрования/MAC, для которой такая комбинация станет небезопасной. Такой подход «все или ничего» уменьшает вероятность недостатков имплементации. В частности, схема аутентифицированного шифрования должна быть имплементирована путем обращения к «подпрограмме шифрования» и «подпрограмме аутентификации сообщений», и имплементация тех подпрограмм должна предусматривать возможность внесения изменений в какой-то момент в будущем. (Такое обычно происходит, когда криптографические библиотеки обновляются или стандарты меняются.) Имплементирование подхода, защита которого зависит от того, как его компоненты внедрены (а не от того, какую защиту они обеспечивают), таким образом, становится опасным. Мы настаиваем, что, если подход отклонен, это не значит, что он небезопасен во всех своих реализациях и реализациях своих компонентов; отнюдь, однако, это значит, что любая реализация подхода должна быть проанализирована, и ее безопасность должна быть доказана перед использованием .

Шифрование и аутентификация . Вспомним, что в этом подходе шифрование и аутентификация осуществляются отдельно. При наличии сообщения m, значение передачи составляет (c, t), где $c \leftarrow Enc_{kE}(m)$ и $t \leftarrow Mac_{kM}(m)$.

Такой подход может не достичь даже самого простого уровня стойкости. Чтобы в этом убедиться, обратите внимание, что защищенный MAC не гарантирует какой-либо стойкости, и поэтому возможно для тега MackM (m) упустить информацию о m , которая попадет к перехватчику.

(В качестве тривиального примера, возьмем защищенный MAC, где первый бит тега всегда равен первому биту сообщения.) Итак, подход «шифрование и аутентификация» может выдать схему, которая даже не имеет неотличимых шифрований в присутсвии пеехватчика.

Фактически, подход «шифрование и аутентификация» скорее всего является незащищенным против атак на основе подобранного открытого текста, даже когда реализован со стандартными компонентами (в отличие от выдуманного оппонента из предыдущего пункта). В частности, если используется детерминированный MAC по типу CBC-MAC, тогда тег, который вычисляется на сообщении (для некоторых фиксированных ключей kM), всегда один и тот же. Это позволяет перехватчику идентифицировать, когда одно и то же сообщение отправляется дважды, а значит схема не является CPA-защищенной. Большинство MAC, используемых на практике, являются детерминированными, и это вызывает реальное беспокойство.

Аутентификация, затем шифрование . Вот в первый раз высчитан тег MAC $t \leftarrow Mack_M(m)$; затем m||t защшифрован, и итоговое значение EnckE (m»t)передается. Мы покажем, что эта комбинация также не обязательно выдаст схему аутентифицированного шифрования.

Фактически, мы уже сталкивались с CPA-защищенной схемой шифрования, для которой данный подход является небезопасным; CBC-режим с дополнением строк, описанный в Разделе 3.7.2. (мы предположим далее, что читатель знаком с данным разделом.) Вспомним, что такая схема работает в первую очередь через дополнение строк открытым текстом (который в нашем случае будет m»t) специфическим образом так, что результат кратный длине блока, после чего происходит шифрование посредством CBC-режима. Во время ши фрования, если ошибка в дополнении строк будет обнаружена после осуществления шифрования CBC-режимом, тогда вернется ошибка «плохое дополнение». Касательно «аутентификации, затем шифрования», это значит, что у нас есть целых два источника потенциального провала шифрования: может быть неправильным дополнение строки или тег MAC может не пройти проверку.

Схематически, алгоритм дешифрования Decr в комбинированной схеме работает следующим образом:

$Dec_{kM,kM}(c)$:

1. Вычислим $m\tilde{}:= Dec_{kE}(c)$. Если ошибка дополнения строки будет обнаружена, вернется ошибка «плохое дополнение» и стоп.

2. Разберем $m\tilde{}$как m||t. Если $Vrfy_{kM}(m, t) = 1$, вернется m; иначе, выведется «провал аутентификации».

Предполагая, что атакующий может отличить два сообщения об ошибках, атакующий может применить ту же самую атаку на основании подобранного шифротекста, описанную в Разделе 3.7.2, к вышеуказанной схеме, чтобы извлечь весь ори-

гинальный открытый текст из данного шифротекста. (Так случится из-за того, что атака на оракула дополнения, показанная в Разделе 3.7.2, опирается на возможность узнать, была и ошибка дополнения, вот что может открыть данная схема.) Такой тип атаки работает успешно на практике в различных условиях, например, в конфигурации IPsec, которая использует метод «аутентификация, затем шифрование».

Один из способов исправить вышеуказанную схему - это обеспечение того, чтобы только одно сообщение об ошибке возвращалось, независимо от того, какая ошибка вызвала сбой. Это является неудовлетворительным решением по нескольким причинам: (1) на это могут быть уважительные причины (например, практичность, отладка) множественных сообщений об ошибках; (2) форсирование одинаковых сообщений об ошибках означает, что комбинация более не является истинно типичной, то есть она требует от внедряющего по методу «аутентификация, затем шифрование» следить за тем, какое сообщение об ошибке возвращается соответствующей CPA-защищенной схемой шифрования; (3) чаще всего, это необычайно трудно обеспечить, чтобы различные ошибки не были распознаны, поскольку, например, разница во времени возвращения каждой из этих ошибок может быть использована злоумышленником для распознавания ошибок (сравните с ранним описанием атак, связанных с временем в конце Раздела 4.2). Некоторые версии SSL пытались использовать только одно сообщение об ошибке вместе с подходом «аутентификация, затем шифрование», но атака на оракула дополнения была все же успешно проведена, используя информацию о времени этого типа. В завершении необходимо сказать, что подход «аутентификация, затем шифрование», в общем, не обеспечивает аутентифицированного шифрования и не может использоваться.

Шифрование, затем аутентификация . В этом подходе сообщение сначала шифруется, а затем MAC вычисляется по результату. То есть сообщение - это пара (c, t) , где $c \leftarrow Enc_{kE}(m)$ и $t \leftarrow Mac_{kM}(c)$.

Дешифрование (c, t) осуществляется путем выведения $\perp$, если VrfykM (c, t) $f\neq$ 1, иначе выводится DeckE (c). См. Конструкцию 4.18 для формального описания

КОНСТРУКЦИЯ 4.18

Пусть Π_E = (Enc, Dec) будет схемой шифрования с закрытым ключом, и пусть Π_M = (Mac, Vrfy) будет кодом аутентификации сообщений, где в каждом случае генерирование ключа выполняется путем простого подбора универсального n-битного ключа. Определим схему шифрования с закрытым ключом (Genr , Encr , Decr) следующим образом:

• Gen^r: при вводе 1^n, выбрать независимый, универсальный kE, kM $\in \{0, 1\}^n$ и вывести ключ (kE, kM).

• Enc^r: при вводе ключ (kE, kM) и открытый текст сообщения m, вычислить $c \leftarrow Enc_{kE}(m)$ и $t \leftarrow Mac_{kM}(c)$. Вывести шифротекст (c, t).

• Dec^r: при вводе ключ (kE, kM) и шифротекст (c, t), сначала проверить верно ли $Vrfy_{kM}(c, t) =? 1$. Если да, вывести $Dec_{kE}(c)$; если нет, тогда вывести $\perp$.

Типичная конструкция схемы аутентифицированного шифрования .

Данный подход *является* работоспособным, пока MAC является строго защищенным, кк определено Определением 4.3. Интуиция касательно защиты данного подхода подсказывает, что шифротекст (c, t) является действительным, если t является действительным тегом MAC на c. Сильная защита MAC обеспечивает то, что злоумышленник не сможет сгенерировать какой-либо верный шифртекст, который он не получил из его оракула шифрования. Это непосредственно предполагает, что Конструкция 4.18 не поддается подделке. Что касается CCA-защиты, MAC, вычисляемый по шифротексту, влияет на оракула шифрования, делая его бесполезным, поскольку за каждым шифротекстом (c, t) злоумышленник обращается к своему оракулу дешифрования либо злоумышленник уже знает дешифрование (если он получил (c, t) от своего собственного оракула шифрования) либо еще как-нибудь может ожидать, что в результате будет ошибка (поскольку злоумышленник не может гене рировать какие-либо новые шифротексты). Это означает, что CCA-защита комбинированной схемы уменьшается до CPA-защиты ПЕ . Можно также увидеть, что MAC проверяется перед дешифрованием ; таким образом, проверка MAC не может упустить каких-либо данных об открытом тексте (в сравнении с атакой с оракулом дополнения, которую мы наблюдали при подходе «аутентификация, затем шифрование»). А теперь мы формализуем вышеперечисленные аргументы.

ТЕОРЕМА 4.19 *Пусть ПЕ будет CPA-защищенной схемой шифрования с закрытым ключом, и пусть ПМ будет строго защищенным кодом аутентификации сообщений. Тогда Конструкция 4.18 является схемой аутентифицированного шифрования.*

ДОКАЗАТЕЛЬСТВО Пусть Π^r обозначает сзему, проистекающую из Конструкции 4.18. Нам необходимо показать, что Π^r не поддается подделке и является защищенной с точки зрения атаки на основе подобранного шифротекста. Согласно представлению, описанному выше, скажем, шифротекст (c, t) является действительным (касательно какого-то фиксированного секретного ключа (kE, kM)), если $Vrfy_{kM}$ (c, t) = 1. Мы покажем, что строгая защита Π_M подразумевает, что (кроме как с пренебрежимо малой вероятностью) любые «новые» шифротексты , которые злоумышленник отправит оракулу дешифрования будут недействительны.

Как было сказано ранее, это непосредственно подразумевает невозможность подделки. (Фактически, это даже сильнее, чем невозможность подделки.) Этот факт также делает оракула дешифрования бесполезным, и ээто значит, что CCA-защита $\Pi^r = (Gen^r, Enc^r, Dec^r)$ уменьшается до CPA-защиты Π_E .

Если говорить более детально, пусть A будет вероятностным полиномиальновременным злоумышленником. атакующим Конструкцию 4.18 атакой на основе подобранного шифротекста (сравните с Определением 3.33). Скажем, шифротекст (c, t) является новым, если A не получил (c, t) от своего оракула шифрования и в качестве шифротекста-испытания. Пусть ValidQuery будет фактом, что

A отправил новый шифротекст (c, t) своему оракулу дешифрования, который действительный, то есть для которого $Vrfy_{kM}$ (c, t) = 1. Мы доказываем:

УТВЕРЖДЕНИЕ 4.20 Pr[ValidQuery] является пренебрежимо малым.

ДОКАЗАТЕЛЬСТВО Интуитивно, это благодаря тому факту, что, если ValidQuery слечается, тогда злоумышленник подделывает новую действителную пару (c, t) в эксперименте Mac-sforge. Строго говоря, пусть q(•) будет полиномиальной верней границей количества запрсов оракулу дешифрования, выполненных A, и рассмотрим следующего злоумышленника AM, атакующего код аутентификации сообщений ПМ (то есть запущенный в эксперименте $Mac\text{-}sforge_{AM.ПМ}(n)$):

Злоумышленник A_M :

A_M получает входные данные 1^n и имеет доступ к оракулу MAC $Mac_{kM}(•)$.

1. Выбираем универсальный $k_E \in \{0, 1\}^n$ и $i \in \{1, \ldots, q(n)\}$.

2. Запускаем A на вводе 1n. Когда A совершает запрос оракулу дешифрования на сообщение m, ответим ему следующим образом:

(i) Вычислите $c \leftarrow Enc_{kE}$ (m).

(ii) Опросите c оракула MAC и получите t в ответ. Верните (c, t) к A.

Шифротекст-испытание приготовлен точно таким же образом (с универсальным битом $b \in \{0, 1\}$, подобранным для выбора сообщения m_b , которое становится зашифрованным).

Когда A совершает запрос оракулу дешифрования касательно шифротекста (c, t), ответим ему следующим образом: Если это запрос i оракулу дешифрования, выведите (c, t). Иначе:

(i) Если (c, t) был ответом на предыдущий запрос оракулу шифрования касательно сообщения m, верните m.

(ii) Иначе, верните $\perp$.

По сути, AM «гадает», что запрос i оракулу дешифрования A будет новым, действующим запросом, который выполняет A . В том случае, AM выводит действительную подделку сообщения c, которое он никогда ранее не передавал своему собственному оракулу MAC. Очевидно, что AM работает в течение вероятностного полиномиального времени. Теперь мы проанализируем вероятность того, что AM выполнил хорошую подделку. Важно то, что видимость A , когда он работает в качестве подпрограммы AM распространяется идентично виду A в эксперименте $\mathsf{PrivK}^{cca}_{\mathcal{A},\Pi}(n)$, пока событие ValidQuery не случится. Чтобы в этом убедиться, обратите внимание, что запрос A оракулу шифрования (равно как и вычисление шифротекста-испытания) идеально симулируется AM . Что касается запроса A оракулу дешифрования, пока не случится ValidQuery, все это останется свойством симуляции. В случае (i) - это очевидно. Что касается случая (ii), если

шифротекст (c, t) , передаваемый оракулу дешифрования, новый, тогда, пока не случился ValidQuery , правильный ответ на запрос оракулу шифрования будет действительно⊥. (Обратите внимание, что случай (i) как раз тот случай, когда (c, t) не новый, а случай (ii) как раз тот случай когда (c, t) новый.) Вспомним, что A не разрешается передавать шифротекст-испытание оракулу дешифрования.

Потому что видимость A когда он работает в качестве подпрограммы AM распространяется идентично видимости A в эксперименте PrivKcca t (n), пока не случится событие ValidQuery, вероятность события ValidQuery в эксперименте Mac-forge идентичная вероятности события в эксперименте $\mathsf{PrivK}^{\mathsf{cca}}_{\mathcal{A},\Pi}(n)$

Если AM правильно угадает первый индекс i , когда случится ValidQuery, тогда AM выведет (c, t), для которого VrfykM (c, t) = 1 (поскольку (c, t) является действительным), и которому никогда не был дан тег t в ответ на запрос MackM (c) (поскольку(c, t) является новым). В этом случае AM достигнет успеха в эксперименте Mac-sforge (n).

Вероятность того, что AM угадает i правильно - 1/q(n). Таким образом, Pr[Mac-sforge $_{\text{AMПМ}}$ (n) = 1] ≥ Pr[ValidQuery]/q(n).

Поскольку ПМ является строго защищенным MAC, a q является полиномиальным, мы в заключении скажем, что Pr[ValidQuery] является незначительным .Мы используем Утверждение 4.20, чтобы доказатель безопасность Пr. Более легкий пример должен доказать, что Π^{r} не поддается подделке. Это вытекает прямо из утверждения, и так мы только что представили скорее неформальное обоснование, чем формальное доказательство. Очевидно, что злоумышленник Ar в эксперименте на шифрование, которое не поддается подделке, является ограниченной версией злоумышленника в эксперименте на подобранный шифротекст (в первом злоумышленник имеет только доступ к оракулу шифрования). Когда Ar выводит шифротекст (c, t) в конце своего эксперимента, он «добивается успеха, только, если (c, t) действительный и новый. Но предыдущее утверждение четко показывает, что вероятность такого события пренебрежимо мала.

Совсем немного сложнее доказать, что Пr является защищенным с точки зрения атаки на основе подобранного шифрованного текста Пусть A снова будет вероятностным полиномиально-временным злоумышленником, атакующим Пr атакой на сонове подобранного шифротекста. Мы имеем

Pr[$\mathsf{PrivK}^{\mathsf{cca}}_{\mathcal{A},\Pi}(n)$=1]A≤Pr[ValidQuery]+ Pr[$\mathsf{PrivK}^{\mathsf{cca}}_{\mathcal{A},\Pi}(n)$= 1^ alidQuery].(4.8)

Мы уже показали, что Pr[ValidQuery] является пренебрежимо малым. Следующее утверждение, таким образом, завершает доказательство теоремы.

УТВЕРЖДЕНИЕ 4.21 Существует пренебрежимо малая функция negl , заключающаяся в том, что

$$\Pr[\mathsf{PrivK}^{\mathsf{cca}}_{\mathcal{A},\Pi'}(n) = 1 \wedge \overline{\mathsf{ValidQuery}}] \leq \frac{1}{2} + \mathsf{negl}(n).$$

Чтобы доказать данное утверждение мы воспользуемся защитой от атак на основе подобранного открытого текста Π_E. Рассмотрим следующего злоумышленника A_E, атакующего Π_E при помощи атаки на основе открытого текста:

Злоумышленник A_E:

A_E получает входные данные 1^n и доступ к $Enc_k(\bullet)$.

1. Выберем универсальный $kM \in \{0, 1\}^n$.

2. Запускаем A на вводе 1n. Когда A совершает запрос оракулу дешифрования на сообщение m, ответим ему следующим образом:

(i) Опрашиваем m у EnckE (•) и получаем c в ответ.

(ii) Вычисляем $t \leftarrow Mac_{kM}$ (c) и возвращаем (c, t) to A.

Когда A совершает запрос оракулу дешифрования на шифротекст (c, t), ответим ему следующим образом:

• Если (c, t) был ответом на предыдущий запрос оракулу шифрования на сообщение m, верните m. Иначе, верните $\perp$.

3. Когда A выводит сообщения (m0, m1), выведите эти же самые сообщения и получите шифротекст-испытаниес в ответ. Вычислите $t \leftarrow$ MackM (c) и верните (c, t) в качестве шифротекста-испытания для A. Продолжайте отвечать на запросы оракула A, как указано выше.

4. Выведите один и тот же бит b^r, что вывел A.

Обратите внимание, что A_E не требует оракула дешифрования, потому что он просто предполагает, сто любой запрос дешифрования A, который не был результатом предыдущего запроса оракулу шифрования является недействительным.

Очевидно, что A_E работает в вероятностном полиномиальном времени. Более того, the view of A видимость A когда он работает в качестве подпрограммы A_E распространяется идентично видимости A в эксперименте $\mathsf{PrivK}^{\mathsf{cca}}_{\mathcal{A},\Pi}(n)$ так как событие ValidQuery ни разу не случилось. Таким образом, вероятность того, что AE добьется успеха, если ValidQuery не случится равняется вероятности того, что A добьется успеха, если ValidQuery не случится; то есть

$$\Pr[\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A}_E,\Pi_E}(n) = 1 \wedge \overline{\mathsf{ValidQuery}}] = \Pr[\mathsf{PrivK}^{\mathsf{cca}}_{\mathcal{A},\Pi'}(n) = 1 \wedge \overline{\mathsf{ValidQuery}}],$$

это значит, что

$$\begin{aligned}\Pr[\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A}_E,\Pi_E}(n) = 1] &\geq \Pr[\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A}_E,\Pi_E}(n) = 1 \wedge \overline{\mathsf{ValidQuery}}] \\ &= \Pr[\mathsf{PrivK}^{\mathsf{cca}}_{\mathcal{A},\Pi'}(n) = 1 \wedge \overline{\mathsf{ValidQuery}}].\end{aligned}$$

Поскольку Π_E является CPA-защищенным, существует пренебрежимо малая функция negl такая, что $\Pr[\mathsf{PrivK}^{\mathsf{cca}}_{\mathcal{A},\Pi}(n) = 1] \leq 1/2 + \mathrm{negl}(n)$. Это доказывает утверждение.

Необходимость внезависимых ключах. Завершим эту главу, сосредоточив внимание на базовом принципе криптографии: различные варианты криптографических примитивов должны всегда использовать независимые ключи. Чтобы проиллюстрировать это здесь, рассмотрим, что может случится с методом «шифрование, затем аутентификация», когда один и тот же ключ k используется как для шифрования, так и для аутентификации. Пусть F будет сильной псевдослучайной перестановкой. Из этого следует, что F^{-1} также является сильной псевдослучайной перестановкой. Определим Enck (m) = Fk (m‖r) для $m \in \{0, 1\}^{n/2}$ и универсальный $r \in \{0, 1\}^{n/2}$, а также определим Mac^k (c) = F_k^{-1}(c). Можно показать, что схема шифрования является безопасной для атак на основе подобранного открытого текста (в действительности, она даже безопасна от атак на основе подобранного шифротекста; см. Упражнение 4.25), а также мы знаем, что данный код аутентификации сообщений является защищенным MAC. Однако, комбинация «шифрование, потом аутентификация», используя один ключ k применимо к сообщению m выдает:

$$Enc_k (m), Mac_k (Enc_k (m)) = F_k (m‖r), F^{-1}(F_k (m‖r)) = F_k (m‖r), m‖r,$$

и сообщение m is revealed in the clear! Это ни в коем случае не противоречит Теореме 4.19, так как Конструкция 4.18 явно требует, чтобы k_M, k_E было подобрано (единообразно и) независимо. Мы даем читателю возможность выяснить, где такая независимость применяется в доказательстве Теоремы 4.19.

4.5.3 Защищенные сеансы связи

Мы кратко описывали применение аутентифицированного шифрования к параметрам двух сторон, желающих осуществлять безопасную связь, а именно, с взаимной стойкостью и целостностью во время сеансов связи. (В этом разделе сеанс связи будет означать простой период времени, во время которого сообщающиеся стороны сохраняют состояние.) В нашем решении мы будем намеренно неформальны; формальное определение достаточно сложное, а данная тема лежит больше в области сетевой безопасности чем криптографии.

Пусть П = (Enc, Dec) будет схемой аутентифицированного шифрования. Рассмотрим две стороны A и B , которые разделяют ключ k и желают использовать этот ключ, чтобы обезопасить связь между ними во время сеанса. Очевидно, что тут необходимо использовать П: Допустим, сторона A хочет передать сообщение m стороне B, она вычисляет c ← Enck (m) и отправляет c стороне B; сторона B в ответ дешифрует c , чтобы извлечь результат (игнорируя результаты, если дешифрирование возвращает ⊥). Аналогично, идентичная процедура следует, когда сторона B хочет отправить сообщение A. Этот простой подход, однако , не удовлетворителен, так как существует множество потенциальных атак:

Атака с пересоставлением Атакующий может поменять местами сообщения. Например, если сторона A передает c1 (шифровку m1) и затем

сразу передает c_2 (шифровку m_2), злоумышленник, у которого есть некоторый доступ к сети, может доставить c_2 перед c_1 , что заставит сторону B вывести сообщение в неправильном порядке. Это приведет к несоответствию просмотров двух сторон своих сеансов связи.

Атака повторного воспроизведения Злоумышленник может воспроизвести (верный) шифротекст c , отправленный ранее оной из сторон. Снова же, это приведет к несоответствию между тем, что одна сторона послала, и что вторая сторона получила.

Атака отражением Злоумышленник может взять шифротекст c , отправленный стороной A стороне B и вернуть его обратно стороне A. И снова же, это вызовет несоответствие между транскрипциями их сенса связи: Сторона A может вывести сообщение m, даже если сторона B никогда не отсылала такого сообщения.

К счастью, вышеописанные атаки легко предотвратить используя счетчики, чтобы решить первые две проблемы и бит направленности, чтобы предотвратить третью. 5 Мы опишем это последовательно. Каждая сторона имеет два счетчика ctrA,B и ctrB,A , отслеживающих количество сообщений, отправленных стороной A стороне B (и наоборот, стороной B стороне A) во время сеанса. Эти счетчики инициализируются на отметке 0 и увеличиваются каждый раз, когда каждая из сторон отправляет или получает (верное) сообщение. Стороны также договариваются по биту bA,B, и определяют bB,A , чтобы выступить дополнением. (Один из способов это сделать - установить bA,B = 0, если равенство A лексикографически меньше, чем равенство B. Когда сторона A хочет передать сообщение m стороне B, она вычисляет шифротекст c ← Enck (bA,B ||ctrA,B|m) и отправляет c; затем она увеличивает счетчик ctrA,B . По получении c сторона B расшифровывает его; если результат $\perp$, она тут же отклоняет его. Иначе, она разбирает дешифрованное сообщение как b»ctr»m. Если $b = b_{A,B}$ и $ctr = ctr_{A,B}$, тогда сторона B выводит m и увеличивает $ctr_{A,B}$; иначе, сторона B отклоняет сообщение. Вышеописанны е шаги, с соответствующими изменениями, применяются, когда сторона B отправляет сообщение стороне A.

Мы отметим, что, так как стороны могут в любом случае сохранять состояние (а именно, счетчики $ctr_{A,B}$ и $ctr_{B,A}$), то они могли бы с легкостью использовать отслеживающую состояние схему аутентифицированного шифрования Π.

4.5.4 Шифрование против атак на основе подобранного шифротекста

Из определения видно, что любая схема аутентифицированного шифрова

---

[5]На практике, вопрос направленности часто решается простым получением отдельных ключей на каждое направление (то есть стороны использую ключ kA для сообщений, отосланный от A к B, и другой ключ kB для сообщений от B к A).

ния является также защищенной от атак на основе подобранного шифротекста. Может ли там быть CCA-защищенная схема шифрования с закрытым ключом, которую можно подделать? В действительности, могут; см. Упражнение 4.25.

Можно представить приложения, в которых защита от атак на основе подобранного шифротекста нужна, а аутентифицированное шифрование - нет. Одним из примеров должно быть шифрование с закрытым ключом для доставки ключа В качестве конкретного примера, скажем, что сервер дает взломостойкий аппаратный маркер пользователю, куда встроен долгосрочный ключ k. Сервер может выгружать свежий кратковременный ключ kr данному маркеру, предоставляя пользователю $Enc_k(kr)$; пользователь, предположительно, передает этот шифротекст маркеру, который дешифрует его и использует k^r для следующего временного периода. Атака на основе подобранного шифротекста в этих условиях могла бы позволить пользователю узнать k^r, что-то, что пользователь, предположительно, не может сделать. (Обратите внимание, что тут атака оракула дополнения, которая опирается на способность пользователя определять, случилась ли ошибка дешифрования, могла бы потенциально быть осуществлена с относительной легкостью.) С другой стороны, не будет много вреда, если пользователь может генерировать «верный» шифротекст, который заставляет маркер использовать произвольный (независимый) ключ k^{rr} для следующего временного периода. (Коенчно же , это зависит от того, что маркер делает с кратковременным ключом.)

Несмотря на вышесказанное, для шифрования с закрытым ключом большинство «естественных» конструкций CCA-защищенных схем, которые мы знаем, все равно удовлетворяют более сильное определение аутентифицированного шифрования. Иными словами, не существует настоящей причины вообще использовать CCA-защищенную схему, которая даже не схема аутентифицированного шифрования,просто потому что у нас нет никаких конструкций, удовлетворяющих первую схему, которые более эффективны, чем конструкции, достигающие второй схемы.

С концептуальной точки зрения, однако, важно сохранить понятия об отличие защиты от атак с на основе подобранного шифротекста и аутентифицированного шифрования. Что касается вопроса защиты от атак на основе подобранного шифротекста, мы по сути не заинтересованы в целостности сообщений; мы желаем обеспечить приватность даже против активного злоумышленника, который может вмешаться в сеанс связи, когда сообщение идет от отправителя получателю. Для сравнения, что касается аутентифицированного шифрования, мы заинтересованы в двойной цели - стойкости и целостности. Мы акцентируем на этом внимание, потому что в условиях открытого ключа, который мы будем изучать в книге далее, разница между аутентифицированным шифрованием и защитой от атак с подобранным шифротекстом более явная.

4.6 *Инфрмационно-теоретические MAC

В предыдущем разделе мы мы изучили коды аутентификации сообщений с

вычислительной стойкостью, то есть, где предполагаются ограничение на время работы. Вспоминая результаты Главы 2, правильно будет спросить, возможно ли применение аутентификации сообщений в случае присутствия неограниченного злоумышленника. В данном разделе мы покажем, при каких условиях достигается информационно-теоретическая (как альтернатива расчетной) защита.

Первое наблюдение заключается в том, что невозможно достигнуть «идеальной» защиты в этом контексте: а именно, мы не можем надеяться получить код аутентификации сообщений, для которого вероятность того, что злоумышленник выведет верный тег на ранее неаутентифицированное сообщение, составит 0. Причина в том, что злоумышленник может просто гадать верный тег t на любом сообщении, и его догадка будет правильной с вероятностью (как минимум) 1/2|t|, где |t| означает длину тег схемы.

Вышеописанный пример говорит нам о том, чего мы можем надеяться добиться: MAC с тегами длиной |t|, где вероятность подлога максимум 1/2|t| даже для неограниченных злоумышленников. Мы увидим, что этого вполне можно достичь, но только при условии ограничений того, сколько сообщений аутентифицированы честными участниками. Для начала мы определяем информационно-теоретическую защиту для кодов аутентификации сообщений. Начальной точкой должно быть проведение эксперимента $\text{Mac-forge}_{\text{АП}}(n)$ который используется чтобы определить безопасность для вычислительно защищенного MAC (сравните с Опрелелением 4.2), но чтобы сбросить параметр защиты n , и потребовать о том, что $\Pr[\text{Mac-forge}_{\text{АП}}(n)= 1]$ должен быть «маленьким» для всех злоумышленников A (и не только злоумышленников, работающих в полиномиальном времени). Как было сказано выше (и как будет формально доказано в Разделе 4.6.2), однако, такое определение недостижимо. Вернее, информационно-теоретическая защита может быть достигнута только, если мы наложим некоторое ограничение на количество аутентифицированных сообщений честными сторонами. Мы здесь рассмотрим наиболее простое условие, когда стороны аутентифицируют всего лишь одно сообщение. Назовем это одноразовой аутентификацией сообщений. Следующий эксперимент изменяет Mac-forge (n) на основании вышепоказаного описания:

Одноразовый эксперимент по аутентификации сообщений $\text{Mac-forge}^{1\text{-time}}_{\mathcal{A},\Pi}$**:**

1. Ключ k генерируется действующим Gen.

2. Злоумышленник A выводит сообщение mr и получает в ответ тег tr ← Mack (mr).

3. A выводит (m, t).

4. Результат эксперимента - определнн как 1, если и только если (1) $Vrfy_k$ (m, t) = 1 и (2) m f≠ m^r.

ОПРЕДЕЛЕНИЕ 4.22 *Код аутентификации сообщений* П = (Gen, Mac, Vrfy) - *это одноразовый ε-защищенный, или просто ε-защищенный, если для всех (даже неограниченных) злоумышленников A:*

$\Pr[\mathsf{Mac\text{-}forge}^{1\text{-}\mathsf{time}}_{\mathcal{A},\Pi} = 1] \leq \varepsilon$.

4.6.1 Построение информационно-теоретических MAC

В данном разделе мы покажем как построить ε-защищенный MAC на основании строго универсальной функции.6 Затем мы покажем простую конструкцию последней.

Пусть h : K× M → T будет ключевой функцией, первый входной элемент которой будет ключ k ∈ K, а второй входной элемент, которы й будет взят из некоей области M. Как обычно, мы пишем $h_k(m)$ вместо h(k, m). Тогда h является строго универсальной (или попарно-независимой), если для любых двух разных входных элементов m, mr значения $h_k(m)$ и h_k (mr) являются универсальными и независимо распределенными в T , где k является универсальным ключом. Это эквивалентно тому, что вероятность того, что h_k (m), h_k (m^r) возьмут на себя какие-либо определенные значения t, tr, равняется 1/|T |2. То есть:

ОПРЕДЕЛЕНИЕ 4.23 *Функция h : K× M → T яаляется строго универсальной, если для всех разных m, mr ∈ M и всех t, tr ∈ T справедливо, что*

$$\Pr[h_k(m) = t \wedge h_k(m') = t'] = \frac{1}{|\mathcal{T}|^2},$$

где вероятность берется выше универсального выбора k ∈ K.

Вышеописанное должно объяснить конструкцию одноразового кода аутентификации сообщений из любой строго универсальной функции h. Тег t на сообщении m полечен путем вычисления hk (m), где ключ k является универсальным; см. Конструкцию 4.24. Очевидно, даже после того, как злоумышленник видит тег tr := hk (mr) для любого сообщения mr, правильный тег hk(m) для любого другого сообщения m все еще единообразно распространяется в T с точки зрения злоумышленника. Таким образом, злоумышленник не может сделать ничего, кроме как в слепую угадывать тег, и такое гадание будет правильным только с вероятностью 1/|T |.

КОНСТРУКЦИЯ 4.24

Пусть h : K × M → T будет строго универсальной функцией. Определим MAC для сообщения в M следующим образом:

- Gen: подберите универсальный k ∈ K и выведите его в качестве ключа.
- Mac: при вводе ключа k ∈ K и сообщения m ∈ M, выведите тег t := hk (m).
- Vrfy: при вводе ключа k ∈ K сообщения m ∈ M и тега t ∈ T, выведите 1, если и только если t =? h (m). (If m *f*∈ M, затем выведите 0.) k

MAC из любой строго универсальной функции .

6Их часто называют строго универсальными хэш функциями, но в криптографических контекстах термин «хэш» имеет другое значение, которое мы увидим далее в книге.

Вышеописанную конструкцию можно рассматривать как аналогичную Конструкции 4.5. Это потому что строго универсальная функция h идентична случайной функции, так как она определяется только дважды.

ТЕОРЕМА 4.25 *Пусть* $h : K \times M \to T$ *будет строго универсальной функцией. Конструкция 4.24 является* $1/|T|$*-защищенным* MAC *для сообщений в M.*

ДОКАЗАТЕЛЬСТВО Пусть A будет злоумышленником. Как правило, в информационно-теоретических условиях мы можем предположить, что A является детерминированным без потери обобщенности. Поэтому сообщение mr , которое выводит A в начале эксперимента зафиксировано. Более того, пара (m, t), которую A выводит в конце эксперимента, является детерминированной функцией тега tr на mr , которые A получает. Таким образом, мы имеем

$$\begin{aligned}\Pr[\mathsf{Mac\text{-}forge}^{1\text{-}\mathsf{time}}_{\mathcal{A},\Pi} = 1] &= \sum_{t' \in \mathcal{T}} \Pr[\mathsf{Mac\text{-}forge}^{1\text{-}\mathsf{time}}_{\mathcal{A},\Pi} = 1 \ \wedge \ h_k(m') = t'] \\ &= \sum_{\substack{t' \in \mathcal{T} \\ (m,t) := \mathcal{A}(t')}} \Pr[h_k(m) = t \ \wedge \ h_k(m') = t'] \\ &= \sum_{\substack{t' \in \mathcal{T} \\ (m,t) := \mathcal{A}(t')}} \frac{1}{|\mathcal{T}|^2} \ = \ \frac{1}{|\mathcal{T}|}.\end{aligned}$$

Это доказывает теорему.

Теперь мы вернемся к классической конструкции строго универсальной функции. Мы предположим некоторые основные знания об арифметически модульно простым числом ; читатели могут обратиться к Разделу 8.1.1 и 8.1.2 для необходимой правочной информации. Зафиксируйте несколько простых чисел p, и пусть $Z_p \stackrel{\text{def}}{=} \{0, \ldots, p-1\}$. Мы берем как наше пространство сообщений $M = Z_p$; пространство возможных тегов тоже будет $T = Z_p$. Ключ (a, b) состоит из пары элементов из Z_p; таким образом, $K = Z_p \times Z_p$. Определим h как

$h_{a,b}(m) \stackrel{\text{def}}{=} [a \bullet m + b \bmod p]$,

где обозначение [X mod p] относится к редукции целого числа X по модулю p (и поэтому $[X \bmod p] \in Z_p$ всегда).

ТЕОРЕМА 4.26 *Для любого простого числа p функция h является строго универсальной.*

ДОКАЗАТЕЛЬСТВО Зафиксируйте и распознайте m, $m^r \in Z_p$ и любойу t, $tr \in Z_p$. Для которых ключи (a, b) это справедливо, что оба $h_{a,b}(m) = t$ и $h_{a,b}(mr) = tr$? Это справедливо только, если $a \bullet m + b = t \bmod p$ и $a \bullet m^r + b = t^r \bmod p$.

Таким образом, мы имеем два линейных уравнения с двумя неизвестными a, b. Эти оба уравнения выполняются точно, если $a = [(t - tr) \bullet (m - m^r)^{-1} \bmod$

p] и b = [t − a • m mod p]; при этом [(m − m^r)−1 mod p] имеет место, потмоу что m f= m^r , и поэтому m − mr f= 0 mod p. Другими словами, это означает, что для любого m, mr, t, t^r , как показано выше, имеется уникальный ключ (a, b) с $h_{a,b}$(m) = t и $h_{a,b}$(m^r) = t^r. Так как существуют ключи |T | , мы сделали вывод, что вероятность (по выбору ключа) того, что $h_{a,b}$(m) = t и $h_{a,b}$(m^r) = tr является ровно 1/|K| = 1/|T |2 , как и было необходимо .

Параметры Конструкции 4.24 . Мы кратко обсудили параметры Конструкции 4.24, когда она реализуется со строго универсальной функцией, описанной выше, игнорируя факт, что p не в степени 2. Конструкция является 1/|T |-защищенным MAC с тегами длиной log |T |; длина тегов оптимальная по уровню достигнутый защиты.

Пусть M будет каким-то фиксированным пространством сообщений, для которого мы хотим сконструировать одноразовый защищенный MAC. Вышеописанная конструкция дает 1/|M|-защищенный MAC с ключами, которые вдвое длиннее сообщений. Читатель может заметить две проблемы здесь, на противоположном конце спектра. Во-первых, если |M| меньше, чем 1/|M| , вероятность подделки неожиданно большая. С другой стороны, если |M| больше чем 1/|M| , вероятность подделки может ыт ькатастрофической; можно было бы принять (каким-то образом) большую вероятность подделки, если тот уровень безопасности может быть достигнут более короткими ключами. Первая проблема (когда значение |M| маленькое) является легкой для решения простым заложением M в большое пространство сообщений M^r посредством, например, дополнения сообщений нулями. Вторая проблема может также быть решена; см. ссылки в конце данной главы.

4.6.2 Ограничение информационно-теоретических MAC

В данном разделе мы изучим ограничения информационно-теоретической аутентификации сообщений. Мы покажем, что любой 2−n-защищенный MAC должен иметь ключи длиной как минимум 2n. Расширение доказательства показывает, что любой A-time 2−n-защищенный MAC (где защита определяется посредством естественной модификации Определения 4.23) требует ключи длиной как минимум (A+1)•n. Вывод в том, что ни один MAC ключами ограниченной длины не сможет обеспечить информационно-теоретическую защиту при аутентификации неограниченного числа сообщений.

В дальнейшем, мы предположим, что пространство сообщений содержит по крайней мере два сообщения; если нет, смысла в общении вообще нет, и тем более в аутентификации.

ТЕОРЕМА 4.27 *Пусть* Π = (Gen, Mac, Vrfy) *будет 2−n-защищенным* MAC, *где все ключи выведенные* Gen *одинаковой длины. Тогда ключи, выведенные* Gen *должны быть длиной как минимум 2n.*

ДОКАЗАТЕЛЬСТВО Зафиксируйте два разных сообщения m_0, m_1 в пространстве сообщений. Интуиция подсказывает, что должно быть как минимум

2^n p вариантов для тега m_0 (и вообще злоумышленник может угадать его с вероятностью большей, чем 2^{-n}); более того, даже будучи обусловленным значением тега для m_0, должно быть 2^n возможностей для тегаm1 (ии вообще злоумышленник мог бы подделать тег на m_1 с вероятность большей, чем 2^{-n}). Так как каждый ключ определяет теги для m_0 и m_1, это означает, что ключи должны быть как минимум $2^n \times 2^n$. Мы сделаем это формальным ниже.

Пусть K обозначает пространство ключей (то есть набор всех возможных ключей, которые могут быть выведены Gen). Для любого возможного тега t0, пусть K(t0) означает набор ключей, для которых t0 является верным тегом на m0; то есть

$K(t_0) \stackrel{\text{def}}{=} \{k \mid \text{Vrfy}_k(m_0, t_0) = 1\}$.

Для любого t_0 мы должны иметь $|K(t_0)| \leq 2^{-n} \bullet |K|$. Иначе злоумышленник может просто вывести (m_0, t_0) в качестве его подлог; это будет верный подлог с вероятностью как минимум $|K(t0)|/|K| > 2-n$, что противоречит заявленной защите.

Рассмотрим теперь злоумышленника A , который запрашивает тег на сообщение m0, получает в ответ тег t0, подбирает универсальный ключ $k \in K(t_0)$ и выводит $(m_1, \text{Mac}_k(m_1))$ в качестве его подлога. Вероятность того, что A выведет верный подлог как минимум

$$\sum_{t_0} \Pr[\mathsf{Mac}_k(m_0) = t_0] \cdot \frac{1}{|\mathcal{K}(m_0, t_0)|} \geq \sum_{t_0} \Pr[\mathsf{Mac}_k(m_0) = t_0] \cdot \frac{2^n}{|\mathcal{K}|}$$
$$= \frac{2^n}{|\mathcal{K}|}.$$

При заявленой защите схемы вероятность того, что злоумышленник выведет верный подлог максимум 2^{-n}. Таким образом, мы должны иметь $|K| \geq 2^{2n}$. Так как все ключи имеют одинаковую длину, каждый ключ должен иметь длину как минимум 2^n.

Ссылки и дополнительная литература

Определение защиты кодов аутентификации сообщений было принято Белларом (Bellare) и др. [18] из определения защиты цифровых подписей, данной Голдвассером (Goldwasser) и др. [81] (см. Главу 12). Для получения дополнительной информации касательно варианта определения, при котором разрешены подтверждающие запросы, см. [17].

Парадигма использования псевдослучайных функций для аутентификации сообщений (как в Контсрукции 4.5) была представлена Голдрайхом (Goldreich) и др. [77]. Конструкция 4.7 разработана Голдрайхом [76].

CBC-MAC был стандартизирован в начале 80-х годов [94, 178] и все еще широко используется. Защищенность (для аутентификации сообщений фиксированной длины) базового CBC-MAC была доказана Белларом и др. [18]. Бернштейн (Bernstein) [26, 27] дает более прямое (хотя, возможно, менее наглядное) доказательство, а также описывает некоторые обобщенные версии CBC-MAC.

Как сказано в данной главе, базовый CBC-MAC является незащищенным при использовании для аутентификации сообщений различной длины. Одним из способов это исправить - это добавить длину к сообщению. Недостаток заключается в невозможности справиться в потоковыми данными, когда длина сообщения неизвестна заранее. Петранк (Petrank) и Рэкофф (Rackoff) [137] предлагают альтернативу, «онлайн» подход решения данного вопроса. Дальнейшие разработки была даны Блэком (Black) и Рогавэем (Rogaway) [32] и Иватой (Iwata) и Куросавой (Kurosawa); это привело к новому предложенному стандарту под названием CMAC.

Важность аутентифицированого шифрования по началу была точно выделена в [100, 19], который предлагает определения идентичные тем, которые мы дали здесь. Беллар и Нампремпрэ (Namprempre) [19] анализируют три типичных подхода, описанных здесь, хотя идея использования подхода «шифрование, затем аутентификация» для достижения защиты от атак на соревновании подобранного шифротекста относится к более раннему периоду, по крайней мере к работам Долева (Dolev) и др. [61]. Кравчик (Krawczyk) [108] проверяет другие методы достижения стойкости и аутентификации, а также анализирует подход «аутентификация, затем шифрование», используемый SSL. Дегабриэль (Degabriele) и Патерсон (Paterson) [54] показывают атаку на IPsec при конфигурации «аутентификация, затем шифрование» (подход по умолчанию для аутентифицированного шифрования, как правило, «шифрование, затем аутентификация»); однако достичь подхода «аутентификация, затем шифрование» можно в некоторых конфигурациях). Некоторые нетипичные схемы для аутентифицированного шифрования также были предложены; см. [110] для подробного сравнения.

Информационно-теоретические MAC впервые были изучены Гилбертом (Gilbert) и др. [73]. Уэгман (Wegman) и Картер (Carter) [177] представили понятие строго универсальных функций и отметили их применение к аутентификации одноразовых сообщений. Они также показывают, как уменьшить длину ключа для этой цели, используя почти строго универсальную функцию. В частности, конструкция, которую мы приводим здесь, достигает 2^{-n}- защиты для сообщений длиной n с ключми длиной O(n); Уэгман и Картер показывают, как построить 2^{-n}-защищенный MAC для сообщений длиной A с ключами (фактически) длоной $O(n \cdot \log A)$. Простая конструкция строго универсальной функции, которую мы приводим здесь, разработана (с небольшими изменениями) картером и Уэгманом [42]. Читатель, заинтересованный в более глубоком изучении информационно-теоретических MAC может обратиться к докладу Стинсона (Stinson) [166], исследованию Симмонса (Simmons) [162] или первой редации пособия Стинсона [167, Глава 10].

Упражнения

4.1 Скажем Π = (Gen, Mac, Vrfy) является защищенным MAC, и для $k \in \{0, 1\}^n$ алгоритм генерирования тегов Mack всегда выводит теги длиной t(n). До-

кажите, что t должен быть суперлогарифмическим или эквивалентно, что если t(n) = O(log n), тогда П не может быть защищенным MAC.

Подсказка: Рассмотрите возможность случайного угадывания верного тега .

4.2 Рассмотрите расширение определения защищенной аутентификации сообщений, при которой злоумышленник обеспечен обеими оракулами, Mac и Vrfy.

(a) Предоставьте формальное объяснение защиты для данного случая.

(b) Предположим, что П является детерминированным MAC, использующим каноническую проверку, которая соответвтует Определению 4.2. Докажите, что П также соответствует вашему определению из части (a).

4.3 Предположим, что имеют место защищенные MAC. Дайте конструкцию MAC, который защищенный согласно Определению 4.2, но незащищенный в случае, если злоумышленник дополнительно имеет доступ к оракулу Vrfy (сравните с предыдущим упражнением).

4.4 Докажите Предположение 4.4.

4.5 Предположим, что имеют место защищенные MAC. Докажите, что имеет место MAC, который защищен (по Определению 4.2), но не строго защищен (по Определению 4.3).

4.6 Рассмотрите следующий MAC для сообщений длиной $A(n) = 2^{n-2}$ с использованием псевдослучайной функции F : При вводе сообщение $m_0 \| m_1$ (при $|m_0| = |m_1| = n - 1$) и ключ $k \in \{0, 1\}^n$, алгоритм Mac выводит $t = F_k (0 \| m_0) \| F_k (1 \| m_1)$. Алгоритм Vrfy определяется естесственнм путем. Защищен ли (Gen, Mac, Vrfy)? Докажите свой ответ.

4.7 Пусть F — псевдослучайная функция. Покажите, что каждый из следующих MAC является незащищенным, даже если используется для аутентификации сообщений фиксированной длины. (В каждом случае Gen выводит универсальный $k \in \{0, 1\}^n$. Пусть (i) обозначает n/2-битное шифрование целого числаi.)

(a) Чтобы аутентифицировать сообщение m = m1, . . . , mA , в котором mi ∈ {0, 1}n, вычислите t := Fk (m1) ⊕ • • • ⊕ Fk (mA).

(b) Чтобы аутентифицировать сообщение m = m1, . . . , mA, в котором mi ∈ {0, 1}n/2, вычислите t := Fk ((1)»m1) ⊕ • • • ⊕ Fk ((A)»mA).

(c) Чтобы аутентифицировать сообщение $m = m_1, \ldots, m_A$, в котором $mi \in \{0, 1\}^{n/2}$, подберите универсальный $r \leftarrow \{0, 1\}^n$, вычислите $t := F_k (r) \oplus F_k ((1) \| m_1) \oplus \cdots \oplus F_k ((A) \| m_A)$, и пусть тег будет (r, t).

4.8 Пусть F будет псевдослучайной функцией. Покажите, что следующий MAC для сообщений длиной 2n является незащищенным: Gen выводит $k \in \{0, 1\}^n$. Чтобы аутентифицировать сообщение $m_1 \| m_2$ с $|m_1| = |m_2| = n$, вычислите тег $Fk (m_1) \| F_k (F_k (m_2))$.

4.9 Давая любой детерминированный MAC (Mac, Vrfy), мы можем наблюдать Mac в качестве ключевой функции. В обеих Конструкциях, 4.5 и 4.11, Mac является псевдослучайно функцией. Дайте конструкцию защищенного, детерминированного MAC, в котором Mac не является псевдослучайно функцией.

4.10 Является ли Конструкция 4.5 достаточно надежной при реализации с использованием слабой псевдослучайной функции (сравните с Упр ажнением 3.26)? Объ ясните.

4.11 Докажите, что Конструкция 4.7 является надежной , даже е сли злоумышленник дополнительно имеет доступ к оракулу Vrfy (сравните с Упражнением 4.2).

4.12 Докажите, что Конструкция 4.7 является надежной , если она изменяется следующим образом: Установите ti := Fk (r»b»i»mi), где b является одиночным битом, таким, что b = 0 во всех блоках, кроме последнего, и b = 1 в последнем блоке. (Предположим для простоты, что длина всех сообщений, подлежащих аутентификации, всегда целое число кратное n/2 − 1.) Какое преимущество такого изменения?

4.13 Мы выясним, что случится, если конструкция базового CBC-MAC используется с сообщениями различной длины.

(a) Скажем, отправитель и получатель не договорились о длине сообщений заранее (и поэтому Vrfy (m, t) = 1 iff t =? Mac (m), независимо от длины m), но отправитель проверяет подлинность только сообщений длиной 2^n. Покажите, что злоумышленник может подделать верный тег на сообщении длиной 4^n.

(b) Скажем, получатель получает 3-блочные сообщения (поэтому Vrfyk (m, t) =1 только, если m имеет длину 3n и t =? Mack (m)), но о тправитель аутентифицирует сообщения любой длины кратной n. Покажите, что злоумышленник может подделать верный тег на новое сообщение.

4.14 Докажите, что следующие изменения базового CBC-MA не дают защищенный MAC (даже для сообщений фиксированной дины):

(a) Mac выводит все блоки $t_1, \ldots, t_A$, а не просто t_A . (Проверка только лишь подтверждает правильность t_A.)

(b) Случайный начальный блок используется каждый раз при аутентификации сообщения. То есть подберите универсальный $t_0 \in \{0, 1\}^n$, запустите бащовый CBC-MAC над «сообщением» $t_0, m_1, \ldots, m_A$ и выведите тег (t_0, t_A). Проверка произведена естественным путем.

4.15 Покажите, что, присоединяя длину сообщения к концу сообщения перед применением базового CBC-MAC, не приводит к защищенности MAC для сообщений проивольной длины.

4.16 Покажите, что шифрование для сообщений произвольной длины, описанное в Разделе 4.4.2 является безпрефиксным.

4.17 Рассмотрите следующее шифрование, обрабатывающее сообщения длиной меньшей, чем n • 2n: Мы шифруем строку m $\in \{0, 1\}^*$, для начала добавляя столько нулей, сколько нужно, чтобы получить дину получившейся строки m отличной от нуля и кратной n. Затем мы добавляем число blocks к m (эквивалентно, добавляем целое число |m |/n), зашифрованное как n-битная строка. Покажите, что такое это шифрование не явля ется безпрефиксным.

4.18 Докажите, что следующие изменения базового CBC-MAC дают защищенный MAC для сообщений произвольной длины (для простоты, предположим, что длина всех сообщений кратная длине блока). Mack (m) сначала обпределяет $k_A = F_k(A)$, где A - это длина m. В таком случае, тег определяется с использованием CBC-MAC с ключом k_A . Проверка произведена естественным путем.

4.19 Пусть F будет ключевой функцией, которая является защищенным (детерминированным) MAC для сообщений длиной n. (Обратите внимание, что F не обязательно быть псевдослучайной перестановкой.) Покажите, что базовый CBC-MAC не обязательно является защищенным MAC (даже для сообщений фиксированной длины) в случае реализации с F.

4.20 Покажите, что Конструкция 4.7 является строго защищенной.

4.21 Покажите, что Конструкция 4.18 может быть не защищенной от атак на основе подобранного шифротекста, если она реализована с помощью защищенного MAC, которые не является строго защищенным.

4.22 Докажите, что Конструкция 4.18 не поддается подделке, если реализована при помощи любой схемы шифрования (даже не защищенной от атак на основе подобранного открытого текста) и любого защищенного MAC (даже если MAC не является строго защищенным).

4.23 Рассмотрите усиленную версию невозможности подделки (Определение 4.16), где A предоставлен доступ к оракулу шифрования.

(a) Напишите формальное определение для этой версии невозможности подделки.

(b) Докажите, что Конструкция 4.18 соответствует этому боле сильному определению, если ПМ является строго защищенным MAC.

(c) Покажите обратным примером, что Конструкция 4.18 не должна соответствовать этому более сильному определению, если ПМ является защищенным MAC, но не строго защищенным. (Сравните с предыдущим упражнением.)

4.24 Докажите, что подход «аутентификация, затем шифрование», реализованный при помощи любой схемы шифрования, защищенной от атак на основе подобранного открытого текста, и любого защищенного MAC, дает схему шифрования, защищенную от атак на основе подобранного открытого текста, которую невозможно подделать.

4.25 Пусть F будет строго псевдослучайной перестановкой, определите следующую схему шифрования фиксированной длины: При вообще сообщения $m \in \{0,1\}^{n/2}$ и ключа $k \in \{0,1\}^n$, алгоритм Enc подбирает универсальный $r \in \{0,1\}^{n/2}$ и вычисляет $c := Fk(m\|r)$. (См. Упражнение 3.18.) Докажите, что эта схема является защищенной от атак на основе подобранного шифротекста, но не является схемой аутентифицированного шифрования.

4.26 Покажите схему шифрования с закрытым ключем, защищенную от атак на основа подобранного открытого текста, которую невозможно подделать, но которая не явялется защищенной от атак на основе подобранного шифротекста.

4.27 Зафиксируйте $A > 0$ и простое число p. Пусть $K = Z_p A+1$, $M = ZA$ и $T = Zp$. Определимр $h : K \times M \to T$ как

$h_{k0,k1,...,kA}(m_1, \ldots, m_A) = [k_0 + \sum_i k_i m_i \mod p]$ . .

Докажите, что h является строго универсальным.

4.28 Зафиксируйте A, $n > 0$. Пусть $K = \{0,1\}^{A \times n} \times \{0,1\}^A$ (интегрированный как булевое значение $A \times n$ матрица A-бесконечномерный вектор), пусть $M = \{0,1\}^n$, и пусть $T = \{0,1\}^A$. определите $h : K \times M \to T$ как $h_{K,v}(m) = K \bullet m \oplus v$, где все операции выполняются по модулю 2. Докажите, что h является строго универсальным.

4.29 Матрица Теплица K это матрица, в которой $K_{i,j} = K_{i-1,j-1}$, где $i, j > 1$; то есть значения вдоль любой диагонали равны. Поэтому матрица Теплица $A \times n$ (для $A > n$) имеет форму

$$\begin{bmatrix} K_{1,1} & K_{1,2} & K_{1,3} & \cdots & K_{1,n} \\ K_{2,1} & K_{1,1} & K_{1,2} & \cdots & K_{1,n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ K_{\ell,1} & K_{\ell-1,1} & K_{\ell-2,1} & \cdots & K_{\ell-n+1,1} \end{bmatrix}$$

Пусть $K = T^{A \times n} \times \{0,1\}^A$ (где $T_{A \times n}$ означает набор матриц Теплица $A \times n$), пусть $M = \{0,1\}^n$ и пусть $T = \{0,1\}^A$. Определите $h : K \times M \to T$ как $h_{K,v}(m) = K \bullet m \oplus v$, где все операции выполняются по модулю 2. Докажите, что h является строго универсальным. Какое здесь преимущество в сравнении с конструкцией в предыдущем упражнении?

4.30 Определите правильно понятие двухразового ε-защищенного MAC, а также дайте конструкцию, отвечающую вашему определению.

4.31 Пусть $\{h_n : K_n \times \{0,1\}^{0 \bullet n} \to \{0,1\}^n\}^n N$ будет таким, что hn будет строго универсальным для всех n, и пусть F будет псевдослучайной функцией. (Где $K \in Kn$ мы запишем $h_K(\bullet)$ вместо $h_n,K(\bullet)$.) Рассмотрите следующий MAC: $Gen(1^n)$ подбирает универсальный $K \in K^n$ и $k \in \{0,1\}^n$ и выводит (K, k). Чтобы аутентифицировать сообщение $m \in \{0,1\}10 \bullet n$, подберите универсальный $r \in \{0,1\}n$ и выведите $(r, h_K(m) \oplus Fk(r))$. Проверка произведена естественным путем. Докажите, что это дает (вычислительно) защищенный MAC для сообщений длиной 10n.

Глава 5

Хэш-функции и их применения

В этой главе мы представим криптографические хэш-функции и изучим некоторые их применения. На более базовом уровне хэш-функция обеспечивает способ преобразования длинной входной строки в более короткую выходную строку, иногда именуемую дайджест. Первоначальным требованием является предотвращение коллизий или двух входных данных, преобразующихся в один и тот же дайджест. Стойкая к коллизиям хэш-функция обладает множеством применений. Одним из примеров, который мы увидим здесь, будет другой подход, стандартизированный как HMAC, для достижения расширения области для кодов аутентификации сообщений.

Помимо этого, хэш-функции стали повсеместно использоваться в криптографии, особенно в случаях, когда требуются свойства, намного более сильные, чем стойкость к коллизиям. Стало довольно обыденным явлением моделировать криптографические хэш-функции как «абсолютно непредсказуемые» (другими словами случайные оракулы), и мы обсудим данную концепцию, а также противоречия, которые ее окружают, более подробно далее в этой главе. Мы здесь затронем только некоторые из применений модели со случайными оракулами, но столкнемся с нею вновь, когда мы перейдем к условиям криптосистемы с открытым ключом.

Хэш-функции интересны тем, что их можно рассматривать как будто они находятся между средами криптосистем с закрытым и открытым ключом. С одной стороны, как мы видим в Главе 6, они (на практике) построены с использованием методов симметричного ключа, и много канонических применений хэш-функций находятся в условиях симметричных ключей. С теоретической точки зрения, однако, существование стойких к коллизиям хэш-функций оказывается представляет качественно более сильное условие, чем существование шифрования с открытым ключом).

5.1 Определения

Хэш-функции являются простыми функциями, которые принимают входные данные определенной длины и сжимают их в короткие выходные данные фиксированной длины. Классическое использование хэш-функций лежит в структурах данных, где они могут быть использованы для построения ханш-таблиц, обеспечивающих время поиска O(1) при хранении набора элементов. В частности, если диапазон хэш-функции H имеет размер N , тогда элемент x хранится в строке H(x) таблицы размером N . Чтобы извлечь x, ей достаточно вычислить H(x) и перебрать по той строке таблицы, где хранятся элементы. «Хорошая» хэш-функция для этих

целей такая, которая дает несколько коллизий, где коллизия - это пара отличающихся элементов x и xr , для которых H(x) = H(xr); в этом случае мы также говорим, что x и xr сталкиваются. (Когда случается коллизия, два элемента оказываются хранимыми в одной и той же ячейке, квеличивая тем самым время поиска.)

Стойкие к коллизиям хэш-функции по сути идентичные. И снова же, их цель - предотвратить коллизии. Однако, существуют фундаментальные различия. В первую очередь, желание минимизировать коллизии в условиях структур данных становится требованием предотвратить коллизий в условиях криптосистемы. Более того, в контексте структур данных мы можем предположить, что набор элементов данных подбирается независимо от хэш-функции и без какого-либо намерения вызвать коллизии. В контексте криптосистемы наоборот, мы сталкиваемся со злоумышленником, который может выбирать элементы с явной целью вызвать коллизии. Это означает, что стойкие к коллизиям хэш-функции разработать намного сложнее.

5.1.1 Стойкость к коллизиям

Проще говоря, функция H является *стойкой к коллизиям,* если она недостижима для любого вероятностного полиномиально-временного алгоритма поиска коллизий в H. Нас будут интересовать только те хэш-функции, чья область больше, чем их диапазон. В таком случае коллизии должны иметь место, но такие коллизии должно быть тяжело найти.

Проще говоря, мы рассмотрим ключевые хэш-функции. То есть H является функцией с двумя входными параметрами, принимающая в качестве параметра ключ s и строкук x, и выводящая строку $H^s(x) \stackrel{def}{=} H(s, x)$. Требование состоит в том, что коллизию в H^s должно быть тяжело найти из-за ключа s, сгенерированного случайным образом. Существуют по крайней мере два отличия между ключами в этом контексте и ключами, которые мы использовали до этого. Отличие первое в том, что не обязательно все строки соответствуют верным ключам (то есть Hs может не быть определен для определенного s), и, таким образом, ключ s будет скорее всего сгенерирован алгоритмом Gen, чем будет подобран единообразно. Отличие второе, и, возможно, более важное, в том, что этот ключ s (как правило) не держится в секрете, и стойкость к коллизиям необходима даже если злоумышленник овладел s. Чтобы подчеркнуть это, мы будем использовать верхний индекс для ключа и писать H^s, а не H_S.

ОПРЕДЕЛЕНИЕ 5.1 *Хэш-функция (с выходной длиной A) - это пара вероятностных полиномиально-временных алгоритмов (Gen, H) , удовлетворяющих следующие условия:*

• *Gen - это вероятностный алгоритм, который принимает в качестве входных данных параметр защиты 1n и выводит ключ s. Мы допускаем, что 1n подразумевается в s.*

• *H принимает в качестве входных данных ключ s и строку x ∈ {0, 1}* и вы-*

водит строку Hs(x) ∈ {0, 1}$^{A(n)}$ (, где n - это значение параметра защиты, подразумеваемого в s).

Если Hs определяется только для входных данных x ∈ {0, 1}A (^{n}) и A^{r}(^{n}) > A(^{n}), тогда мы скажем, что (Gen, H) = это хэш-функция фиксированной длины для входных данных длиной Ar. В этом случае мы также назовем H функцией сжатия.

В случае с фиксированной длиной нам необходимо, чтобы A^{r} была больше, чем A. Это будет означать, что функция сжала свои входные данные. В общем случае функция принимает в качестве входных данных строки произвольной длины. Таким образом, она также осуществляет сжатия (хотья толькьо строк большей длины, чем A(n)). Обратите внимание, что без сжатия стойкость к коллизиям является тривиальной (так как можно просто взять тождественную функцию H^{s}(x) = x). Теперь мы перейдем к определению защиты. Как правило, сначала мы определим эксперимент для хэш-функции П = (Gen, H), злоумышленника A и параметра защиты n:

Эксперимент по поиску коллизии Hash-collA,П(n):

1. Ключ s генерируется действующим Gen(1n).

2. Злоумышленник A получил s и выводит x, x^{r} . (If П - это хэш-функция фиксированной длины для входных данных длиной Ar(n), затем на потребуется x, xr ∈ {0, 1}$^{A (n)}$.)

3. Вывод эксперимента определенно 1 , если и только если x f = x^{r} и Hs(x) = Hs(xr). В таком случае мы скажем, что A нашел коллизию.

Определение стойкости к коллизиям подразумевает, что ни один умелый злоумышленник не сможет найти коллизию в вышеописанном эксперименте, разве что с пренебрежимо мало вероятностью.

ОПРЕДЕЛЕНИЕ 5.2 *Хэш-функция П = (Gen, H) является стойкой к коллизиям, если для вероятностных полиномиально-временных противников A существует пренебрежимо малая функция negl, такая, что*

Pr [Hash-coll$_{A,П}$(n) = 1] ≤ negl(n).

Для упрощения, мы иногда называем H или Hs как «стойкая к коллизиям хэш-функция», пусть технически мы должны говорить только, что и (Gen, H) тоже. Это не должно воодить в заблуждение.

Криптографические хэш-функции разработаны с одной ясной целью - быть стойкими к коллизиям (помимо прочего). Мы обсудим некоторые реальные хэш-функции в Главе 6. В Разделе 8.4.2 мы увидим, насколько это возможно сконструировать хэш-функции с подтвержденными коллизиями на основании на предположении о сложности определенных теоретико-числовых проблем.

Бесключевые хэш-функции. Криптографические хэш-функции, используемые на практике, в общем, имеют фиксированные выходные данные (также как и блоч-

ные шифры имеют фиксированную длину ключа) и, как правило, являются нелючевыми, что означает, что хэш-функция - это всего лишь фиксиров анная функция $H : \{0, 1\}^* \rightarrow \{0, 1\}^A$. Это проблематично с теоретической точки зрения, так как для любой такой функции всегда существует постоянный алгоритм , который выводит коллизию H: алгоритм просто выводит сталкивающуюся пару (x, x^r), жестко закодированную в сам алгоритм. Использование ключевых хэш-функций решает эту техническую проблему, так как невозможно жестко закодировать сталкивающуюся пару для каждого возможного ключа, используя оправданное количества пространства (и в асимптотических условиях будет невозможно жестко закодировать сталкивающиюся пару для каждого значения параметра защиты).

Несмотря на вышесказанное, (бесключевые) криптографические хэш-функции, используемые на практике, являются устойчивыми к коллизиям для всех практических целей, так как сталкивающиеся пары неизвестны (и их сложно найти вычислительно), даже если они на самом деле имеют место. Доказательство защиты какой-либо конструкции на основе стойкости к коллизиям хэш-функции имеет смысл, даже если используется неключевая хэш-функция H, так как доказательство показывает, что умелый злоумышленник, «ломающий» примитив, может быть использован для того, чтобы легко найти коллизию в H. (Все доказательства в этой книге отвечают этому условию.) В этом случае интерпретация доказательства защиты заключается в том, что, если противник может взломать схему на практике, он может быть использован и для того, чтобы найти коллизию на практике, а это, мы полагаем, не так просто сделать.

5.1.2 Более слабые понятия о защите

В некоторых применениях проще надеяться на требования безопасности, чем на стойкость к коллзиям. Сюда входит :

• Стойкость второго прообраза или стойкость к целевой коллизии: Проще говоря, хэш-функция является стойкой к нахождению второго прообраза, при наличии s и универсального x, это невозможно для ppt злоумышленника найти $x^r \neq x$ так, чтобы $H^s(x^r) = H^s(x)$.

• Стоикость прообраза: Проще говоря, хэш-функция является стойкой к нахождению прообраза, при наличии s и универсального y это невозможно для ppt это невозможно значение x так, чтобы $H^s(x) = y$. (Забегая вперед в Главу 7, это по сути означает, что Hs является односторонним.)

Любая хэш-функция, которая является стойкой к коллизиям, также является стойкой к нахождению второго прообраза. Это справедливо, так как если, при условии получения универсального x, злоумышленник может найти $xr \neq x$, для которого $H^s(x^r) = H^s(x)$, тогда он явно может найти сталкивающуюся пару x и x^r.

Аналогично этому, любая хэш-функция, которая является стойкой к на-

хождению второго прообраза, также является устойчивой к нахождению прообраза. Происходит это из-за того факта, что, если бы это было возможно, имея y, найти x так, чтобы $H^s(x) = y$, тогда можно было бы также взять полученные входные данные xr, вычислить $y := H^s(x^r)$ и затем получить x с Hs(x) = y. С высокой вероятностью xr $\neq$ x (опираясь на факт, что H сжимается, и поэтому множественные входные данные преобразовываются в одни и те же выходные данные), в которых второй прообраз был найден.

Мы, строго говоря, не определяем вышеописанные понятия или доказываем вышеуказанные импликации, так как они не используются дальше в книге. Вам предложат формализировать вышеописанное в Упражнении 5.1.

5.2 Расширение области: Преобразование Меркле-Дамгорда

Конструирование хэш-функций зачастую начинается с разработки стойкой к коллизиям функции сжатия, которая обрабатывает входные данные фиксированной длины, затем используется расширение области, чтобы обработать входные данные произвольной длины. В данном разделе мы покажем одно из решений проблемы расширения области. Мы вернемся к вопросу разработки стойкой к коллизиям функции сжатия в Разделе 6.3.

Преобразование Меркле-Дамгорда является стандартным подходом для расширения функции сжатия до полноценной хэш-функции, которая при этом сохранила свойство стойкости к коллизиям. Он широко используется на практике для хэш-функций, включая семейства MD5 и SHA (см. Раздел 6.3). Существование этого преобразования означает, что при разработке стойких к коллизиям хэш-функций мы можем меньше уделять внимание дынным фиксированной длины. В свою очередь, это делает процесс разработки стойких к коллизиям хэш-функции намного более простым. Преобразование Меркле-Дамгорда также представляет интерес с теоретической точки зрения, так как он подразумевает, что сжатие на один бит настолько же просто (или сложно), как и сжатие на произвольное количество.

Для конкретности, предположим, что функция сжатия (Gen, h) сжимает входные данные в полтора раза; скажем, длина входных данных n , а длина выходных данных n. (Конструкция работает независимо от длины входных/выходных данных до тех пор, пока h сжимает.) Мы построили стойкую к коллизиям хэш-функцию (Gen, H), которая преобразовывает входные данные произвольной длины в выходные данные длиной n. (Gen остается неизменным.) Преобразование Меркле-Дамгорда определено в Конструкции 5.3 и изображено на Рисунке 5.1. Значение z0 , используемое в шаге 2 конструкции под названием вектор инициализации или IV является произвольным и может быть заменено любой константой.

КОНСТРУКЦИЯ 5.3

Пусть (Gen, h) будет хэш-функцией фиксированной длины для входных данных длиной 2n и ывыходных данных длиной n. Постройте хэш-функцию (Gen, H) следующим образом:

• Gen: остается неизменным.

• H: при вводе ключа s и строки $x \in \{0, 1\}^*$ длиной $L < 2^n$, сделайте слудующее:

1. Установите B := [L/n] , (то есть количество блоков в x). Заполните x нулями, чтобы ее длина стала кратной n. Разберите дополненный результат как последовательность n-битных блоков x1, . . . , xB . Установите xB+1 := L, где L зашифрована как n-битная строка.

2. Установите $z_0 := 0^n$. (Это также называется IV.)

3. Для i = 1, . . . , B + 1 вычислите $z_i := hs(zi-1||xi)$.

4. Выведите zB+1.

Преобразование Меркле-Дамгорда.

ТЕОРЕМА 5.4 *Если (Gen, h) является стойкой к коллизиям, тогда (Gen, H) тоже.*

ДОКАЗАТЕЛЬСТВО Мы покажем, что для любых s коллизия в H^s даст коллизию в h^s. Пусть x и x^r будут две различные строки длиной L и Lr, соответственно, так что $H^s(x) = H^s(x^r)$. Пусть $x_1, \ldots, x_B$ будет блоками B дополненной x, и x_1 xz... xB xB+1=L zo = IV hs z1 hs... hszB hs H (x) пусть $x^r, \ldots, x^r$ будет блоками Br дополненной xr. Напомним, что $x_{B+1} = L$ и 1 B_t x^r r $B_{t+1} = L$. Для рассмотрения имеется два случая:

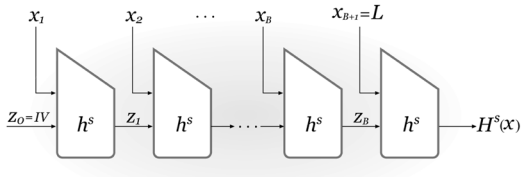

FIGURE 5.1: Преобразование Меркле-Дамгорда.

1. Случай 1: L *f*= L^r. В этом случае последний шаг вычислений

$H^s(x)$ - это $z_{B+1} := h^s(z_B \| L)$, а последний шаг вычислений

$H^s(x^r)$ - это $zr := h(zr \| L^r)$. Так как $H^s(x) = H^s(x^r)$, из этого следует, что hs(zB «L) = hs(zr ||Lr). Однако, L *f*= Lr, и поэтому z_B ||L и z^r $\|L^r$ являются двумя различными строками, которые сталкиваются под h^s.

2. Случай 2: $L = L^r$. Это означает, что $B = B^r$. Пусть $z_0, \ldots, z_{B+1}$ будут значениями, определенными во время вычисления $H^s(x)$, пусть $Ii \stackrel{def}{=} z_{i-1}\|x_i$ обозначает входные данные i для h^s, и установим $I_{B+2} \stackrel{def}{=} z_{B+1}$ Определеите $I^r, \ldots, I^r$ аналогично 1 B+2 относительно x^r. Пусть N будет наибольшим индексом, для

которого I_N *f*= I^r . Поскольку |x| = $|x^r|$, но x *f*= x^r, имеет место i c xi *f*= x^r , и поэтому такой N определенно существует. Потому что

I_{B+2}= z_{B+1} =$H^s(x) = H^s(x^r) = z^r_{B+1}$= I_{B+2}

мы имеем N ≤ B + 1. По максимальности N, мы имеем $I_{N+1} = I^r$ и в частности $z_N = z^r$. Но это означает, что IN , I^r являются коллизией в hs.

Мы оставим это в качестве упражнения для превращения вышесказанного в формальное сжатие.

5.3 Аутентификация сообщений с использованием хэш-функций

В предыдущей главе мы представили конструкции кодов аутентификации сообщений для сообщейний произвольной длины. Первый подход был типичным, но неэффективным. Второй подход, CBC-MAC, основывался на псевдослучайных функциях. Тут мы увидим еще один подход, который мы называем «хэш и MAC» (hash-and-MAC), который опирается на стойкое к коллизиям хэширование вместе с кодом аутентификации сообщений. Затем мы рассмотрим стандартизированную и широко используемую конструкцию под названием HMAC, которую можно считать реализацией данного подхода.

5.3.1 Хэш и MAC

Идея, что кроется под хэш и MAC, проста. Для начала сообщение произвольной длины m хэшируется в строку фиксированной длины Hs(m) при помощи стойкой к коллизиям хэш-функции. Затем MAC (фиксированной длины) применяется к результату. См. Конструкцию 5.5 для формального описания.

КОНСТРУКЦИЯ 5.5

Пусть П = (Mac, Vrfy) будет MAC для сообщений длиной A(n), и пусть ПН = (GenH, H) будет хэш-функцией с длиной выходных данных A(n). Постройте MAC $П^r$ = (Gen^r , Mac^r , $Vrfy^r$) для сообщений произвольной длины следующим образом:

• Gen^r : при вводе 1^n, подберите универсальный k ∈ {0, 1}, и запустите GenH (1n) чтобы получить s; ключ - k^r := (k, s).

• Mac^r : при вводе ключа (k, s) и сообщения m ∈ {0, 1}*, выведите

t ← Mack(H^s(m)).

• $Vrfy^r$: при вводе ключа (k, s), сообщение m ∈ {0, 1}*, и tag t MAC, выведите 1, если и только если Vrfy (H^s(m), t) =? 1.

Парадигма хэш-и-MAC.

Конструкция 5.5 является защищенной, если П является защищенным MAC для сообщений фиксированной длины, и (Gen, H) является стойкой к коллизиям. Очевидно, поскольку хэш-функция является стойкой к коллизиям, аутентификация Hs(m) настолько же эффективна, как и и аутентификация смого m : если отправитель может убедиться в том,

что получатель получит правильное значение Hs(m), стойкость к коллизиям гарантирует, что атакующий не сможет найти отдельное сообщение mr , которое хэшируется в это же значение. Строго говоря, скажем, отправитель использует Конструкцию 5.5, чтобы аутентифицировать некоторый набор сообщений Q, а атакующий A может подделать верный тег на новом сообщении m* $f \in$ Q. Существуют два возможных случая:

Случай 1: *есть сообщение* $m \in Q$ *, так что* $H^s(m^*) = Hs(m)$. Тогда A нашел коллизию в H^s, противоречащую стойкости к коллизиям (Gen, H).

Случай 2: *для каждого сообщения* $m \in Q$ *справедливо, что* $Hs(m^*)\ f = H^s(m)$. Пусть $Hs(Q) \stackrel{def}{=} \{H^s(m) \mid m \in Q\}$. Тогда $Hs(m^*) \notin H^s(Q)$. В этом случае A подделал верный тег на «новом сообщении» Hs(m*) относительно кода аутентификации сообщений фиксированной длины П. Это противоречит предположению, что П является защищенным MAC. Сейчас мы превратим вышесказанное в формальное доказательство.

ТЕОРЕМА 5.6 *Если П является защищенным MAC для сообщений длиной A и ПН являейтся стойкой к коллизиям, тогда Конструкция 5.5* **является защищенным MAC (для сообщений произвольной длины).**

ДОКАЗАТЕЛЬСТВО Пусть $П^r$ обозначает Конструкцию 5.5, и пусть Ar будет ppt злоумышленником, атакующим $П^r$. При выполнении эксперимента Mac-forge $_{A,П}$ t t (n), пусть $k^r = (k, s)$ обозначает ключ MAC, пусть Q обозначает набор сообщений, теги которых были запрошены Ar, и пусть (m* , t) будут конечными выходными данными Ar. Предположим без потери обобщенности, что m* $f \notin$ Q. Определите coll как событие, которое в эксперименте Mac-forge $_{A,П}$ t t (n), существует m $\in$ Q , для которого Hs(m*) = Hs(m). Мы имеем Pr[Mac-forge t t (n) = 1]

$$\begin{aligned}
&\Pr[\mathsf{Mac\text{-}forge}_{\mathcal{A}',\Pi'}(n) = 1] \\
&= \Pr[\mathsf{Mac\text{-}forge}_{\mathcal{A}',\Pi'}(n) = 1 \wedge \mathsf{coll}] + \Pr[\mathsf{Mac\text{-}forge}_{\mathcal{A}',\Pi'}(n) = 1 \wedge \overline{\mathsf{coll}}] \\
&\leq \Pr[\mathsf{coll}] + \Pr[\mathsf{Mac\text{-}forge}_{\mathcal{A}',\Pi'}(n) = 1 \wedge \overline{\mathsf{coll}}]. \qquad (5.1)
\end{aligned}$$

Мы покажем, что оба члена Уравнения (5.1) пренебрежимо малы, таким образом, завершив доказательство. Очевидно, что первый член пренебрежимо мал из-за стойкости к коллизиям ПН, а второй член пренебрежимо мал из-за защиты П.

Рассмотрите следующий алгоритм C на предмет поиска коллизий в ПН :

Алгоритм C:

В алгоритм вводится s в качестве входных данных (с подразумеваемым n).

• Подберите универсальный $k \in \{0, 1\}^n$.

• Запустите $A^r(1^n)$. Когда A^r запросит тег на сообщение i mi $\in \{0, 1\}^*$, вычислите ti $\leftarrow$ Mack (H^s(mi)) и дайте (A^r) (ti).

• Если Ar выведет (m* , t), тогда, если i, для которого Hs(m*) = Hs(mi), вы-

ведите (m* , mi).

Очевидно, что C работает в полиномиальном времени. Давайте проанализируем его поведение. Если входные данные для C генерируются работающим GenH (1^n) для получения s, видимость A^r при работе в качестве подпрограммы C распространяется идентично видимости A^r в эксперименте Mac-forge $_{A,\Pi}$ t t (n). В частности, теги, полученные A^r от C имеют такое же распространение, как и теги, которые A^r получает отMac-forge $_{A,\Pi}$. Поскольку C выводит коллизию сразу же, как случается coll, мы имеем

Pr[Hash-coll$_{C,\Pi H}$ (n) = 1] = Pr[coll].

Так как ПН является стойкой к коллизии, сделаем вывод, что Pr[coll] пренебрежимо мала.

Теперь продолжим доказывать, что второй член в Уравнении (5.1) пренебрежимо малым. Рассмотрите следующего злоумышленника A, атакующего П в Mac-forge $_{A,\Pi}$(n):

Злоумышленник A:

Злоумышленник получил доступ к оракулу MAC Mack (•).

• Вычислите GenH (1n), чтобы получить s.

• Запустите $A^r(1^n)$. Если Ar запрашивает тег на сообщение i mi $\in$ {0, 1}*, тогда: (1) вычислите mˆ i := Hs(mi); (2) получите тег ti на mˆ i от оракула MAC; и (3) дайте (Ar) (ti).

• Если Ar выводитs (m* , t), тогда выведите (H^s(m*), t).

Очевидно, что A работает в полиномиальном времени. Рассмотрите эксперимент Mac-forge $_{A,\Pi}$(n). В том эксперименте, видимость A^r , работающего в качестве подпрограммы A , распространяется идентично его видимости в эксперименте Mac-forge $_{A,\Pi}$t t (n). Более того, даже если и Mac-forge $_{A,\Pi}$t (n) = 1, и coll не случается, A выводит верную подделку. (В таком случае t является верным тегом на H^s(m*) в схеме П относительно k. Факт, что coll не случилась, означает, что H^s(m*) никогда не был опрошен A своему собственному оракулу MAC, и поэтому это действительно подделка.) Таким образом,

Pr[Mac-forge $_{A,\Pi}$ = 1] = Pr[Mac-forge $_{A,\Pi}$ (n) ^ coll],

защита П предполагает, что прежняя вероятность пренебрежимо мала. Это завершает доказателсьтво теоремы.

5.3.2 HMAC

Все конструкции кодов аутентификации сообщений, которые мы видели ранее, так или иначе основаны на каком-то блочном шифре. Существует ли возможность построить защищенный MAC (для сообщений произвольной длины) на непосред-

ственной основе хэш-функции? Первой мыслью является определение Mack (m) = H(k‖m); мы должны предполагать, что, если H является «хорошей» хэш-функцией, тогда должно быть тяжело атакующему предугадать значение H(k‖mr), учитывая значение H(k‖m), для любых mr *f*= m, предполагая, что k подобран случайным образом (и неизвестен атакующему). К несчастью, если H построена с использованием преобразования Меркле-Дамгорда, как и большинство реальных хэш-функций, тогда MAC, разработанный таким образом, является полностью незащищенным, как вам и было предложено показать в Упражнении 5.10.

Вместо этого, мы можем попробовать использовать два слоя хэширования. См. Конструкцию 5.7 для стандартизированной схемы под названием HMAC на основе данной идеи.

КОНСТРУКЦИЯ 5.7

Пусть (Gen_H, H) будет хэш-функцией, сконструированной с применением преобрахзования Меркле-Дамгорда для функции сжатия (Gen_H, h), принимающей входные данные длиной n + n^r. (См. текст.) Пусть opad и ipad будут фиксированными константами длиной n^r. Определяйте MAC следующим образом:

• Gen: при вводе 1n, запустите Gen_H (1^n), чтобы получить ключ s. Также подберите универсальный k ∈ {0, 1 nt . } Выведите ключ (s, k).

• Mac: при вводе ключа (s, k) и сообщения m ∈ {0, 1}* , выведите

t := H^s.(k ⊕ opad) " Hs. (k ⊕ ipad) " m.. .

• Vrfy: при вводе ключа (s, k), сообщения m ∈ {0, 1}* и тега t, выведите 1, если и только если t =? H^s.(k ⊕opad) || H^s. (k ⊕ ipad) || m...

HMAC.

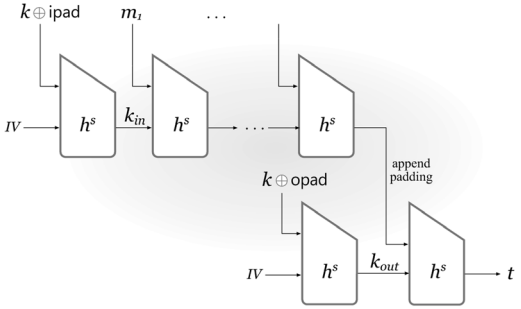

РИСУНОК 5.2: HMAC, наглядный пример.

Почему у нас должна быть какая-то уверенность, что HMAC является защищенным? Одна из причин заключается в том, что мы можем рассматривать HMAC как специфическую реализацию парадигмы хэш-и-MAC из предыдущего раздела. Чтобы это увидеть, мы заглянем «за кулисы» на то, что происходит, когда сообщение проходит аутентификацию; см. Рисунко 5.2. Мы так же

должны более тщательно определять параметры и более подробно остановится на том, как на практике внедряется преобразование Меркле-Дамгорда.

Скажем, (Gen_H, H) сконструирована на основе функции сжатия (GenH, h), в которой h преоразовывает входные данные длиной n + nr в выходные данные длиной n (где, строго говоря, nr - это функция n). Когда мы описывали преобразование Меркле-Дамгорда в Разделе 5.2, мы предполагали, что $n^r = n$, но не обязательно должно быть именно так. Мы также говорили, что длина сообщения, которое должно хэшироваться, была зашифрована в дополнительный блок, который присоединялся к сообщению. На практике длина вместо этого зашифровывается в часть блока с использованием битов $A < n^r$. То есть вычисление Hs(x) начинается с добавления x с нулями к строке длиной ровно на A меньше, чем кратное nr; затем добавляется длина L = |x|, зашифрованная с использованием ровно A бит. Хэш получившейся последовательности nr-битных блоков x1, . . . затем вычисляется как в Конструкции 5.3. Мы предположим, что $n + A \leq nr$. Это означает, в частности, что если мы хэшируем входные данные x длиной $n_r + n$, тогда дополненный результат (включая длину) будет точно длиной 2nr бит. Доказательство Теоремы 5.4, показывающее, что (GenH, H) является стойкой к коллизиям, если (Gen_H, h) стойкая к коллизиям, остается неизменным.

Возвращаясь к HMAC и глядя на Рисунок 5.2, мы можем увидеть, что общая форма HMAC задействует хэширование сообщения произвольной длины в короткую строку $y \stackrel{def}{=} Hs((k \oplus ipad) « m)$, а затем вычисление (секретно кодированной) функции $Hs((k \oplus opad) « y)$ результата. Но мы можем сказать даже еще больше. Сначала обратите внимания, что «внутреннее» вычисление

$$\tilde{H}s(m) \stackrel{def}{=} H^s((k \oplus ipad) \| m)$$

является стойким к коллизиям (предпологая, что h имеет место) при любом значении $k \in ipad$. Более того, первый шаг «внешнего» вычисления $Hs((k \oplus opad) \| y)$ должен вычислить значение $k_{out} \stackrel{def}{=} h^s(IV \| (k \oplus opad))$. Затем, мы определим $h^S(k \| \hat{y})$, где y ссылается на дополненное значение y (то есть включая длину $(k \oplus opad) \| y$, которая всегда $n_r + n$ бит, зашифрованная с использованием ровно A бит). Таким образом, если мы будем считать k_{out} как универсальным, мы будем относиться к нижеописанному более формально и предположим, что

$$\tilde{M}ac_k(y) \stackrel{def}{=} h^s(k \| \hat{y}) \qquad (5.2)$$

является защищенным MAC фиксированной длины, тогда HMAC можно рассиматривать как реализацию подхода хэш-и-MAC с

$$HMAC_{s,k}(m) = \tilde{M}ac_{k_{out}}(\tilde{H}(m)) \qquad (5.3)$$

(где $k_{out} = hs(IV \| (k \oplus opad))$). Благодаря способу разработки функции сжатия h (см. Раздел 6.3.1), предположение, что $\tilde{M}ac$ является защищенным MAC фиксированной длины, имеет смысл.

Роли ipad и opad. Учитывая вышесказанное, можно удивиться, почем необходимо вводить k во «внутреннее» вычисление Hs ((k ⊕ ipad) || m). (В частности, для того, чтобы сделать подход хэш-и-MAC защищенным, нам нужна стойкость к коллизиям на первом шаге, где не требуется секретный ключ.) Причина в том, что это позволяет построить защиту HMAC на основе потенциально более слабого предположения, что (GenH, H) является слабо стойкой к коллизиям, где слабая стойкость к коллизиям определяется следующим экспериментом: ключ s генерируется с использованием GenH, и подбирается универсальный секретный kin ∈ {0, 1}^{n}. Затем злоумышленнику разрешаетс я взаимодействовать с «хэш-оракулом», который возвращает H^s (m) в ответ на запрос m, где H^s ссылается на вычисление H^s с использованием преобразования Меркле-Дамгорда, применимого к h^s, но с использованием секретного значения kin в качестве IV. (Снова см. Рисунок 5.2.) У злоумышленника получится, если он сможет вывести различные значения m, m^r так, чтобы H^s in(m) = Hsin(m^r), и мы скажем, что (Gen_H, H) является слабо стойкой к коллизиям , если у каждого ppt A получится осуществить данный эксперимент только с пренебрежимо малой вероятностью. Если (Gen_H, H) является стойкой к коллизиям, очевидно, что она является слабо стойкой к коллизиям; последнее условие, однако, является наиболее слабым, которое протенциально легче всего удовлетворить. Это хороший пример работающей техники обеспечения защиты. Эта защитная расчетная стратегия окупила себя, когда было обнаружено, что хэш-функция MD5 (см. Раздел 6.3.2) не была стойкой к коллзиям. Атаки нахождения коллизии в MD5 не нарули слабой стойкости к коллизиям, и HMAC-MD5 не был взломан, хотя MD5 взломали. Это дало разработчикам время для замены MD5 на HMAC без сиюминутного страха быть атакованными. (Несмотря на это, HMAC-MD5 не должен быть использован теперь, когда известны недостатки MD5.)

Вышеописанное предполагает, что независимые ключи должны использоваться во внутренних и внешних вычисениях. Ради эффективности единичный ключ k используется в HMAC, но ключ используется в сочетании с ipad и opad , чтобы вывести два других ключа. Определим

$tt^s(k) \stackrel{def}{=} h^s$ .IV || (k ⊕ opad). || h^s .IV || (k⊕ ipad). = k|| k . (5.4)

Если мы предположим, что tts является псевдослучайным генератором для любой s, тогда k_{out} и k_{in} могут быть расценены как независимые и универсальные ключи, если k является универсальным клюом. Защита HMAC в таком случае уменьшается до защиты следующей конструкции:

$$\mathsf{Mac}_{s,k_{in},k_{out}}(m) = h^s\left(k_{out} \parallel H^s_{k_{in}}(m)\right).$$

(Сравните с Уравнением (5.3).) Как было отмечено ранее, данная конструкция может оказаться защищенной (при использовании варианта доказательства для

подхода хэш-и-MAC), если H слабо стойкая от коллизий, и MAC, определенный в Уравнении (5.2) - защищенный MAC фиксированной длины.

ТЕОРЕМА 5.8 *Предположим, что tts , как определено в Уравнении (5.4) является псевдослучайным генератором любых s, MAC, определенный в Уравнении (5.2) - это MAC фиксированной длины для сообщений длиной n, и (GenH, H) является слабо стойкой к коллизиям. В таком случае HMAC является защищенным MAC (для сообщений произвольной длины).*

HMAC на практике . HMAC - это промышленный стандарт, и он ш ироко применяется на практике. Он высокоэффективен и прост для внедрения, а также подкреплен доказательством защиты, основанным на предположениях, которые считаются верными для практических хэш-функций. Своей важностью HMAC частично обязан своевременности своего появления. Перед представлением HMAC многие практикующие специалисты отказывались использовать CBC-MAC (с претензией, что он был «слишком медленный») и вместо него использовали конструкции, которые были незащищены. HMAC предоставил стандартизированный, защищенный способ создания аутентификации сообщений, основанный на хэш-функциях.

5.4 Типичные атаки на хэш-функции

На какую наилучшую защиту для хэш-функции H мы можем рассчитывать? Мы изучили вопрос при помощи демонстрации двух атак, которые являлись типичными в том смысле, что они применялись к произвольным хэш-функциям. Существование этих атак подразумевает, что нижние границы выходной длины H нуждались в достижении некоторого желательного уровня защиты, и, таким образом, имеют важные практические последствия.

5.4.1 Атака «дней рождения» для нахождения коллизий

Пусть $h : \{0, 1\}^* \rightarrow \{0, 1\}^A$ будет хэш-функцией. (Здесь и в сотавшейся части главы мы опустим явное упоминание хэш-ключа s , так как это не совсем относится к делу. Можно рассматривать s как такой, что был сгенерирован и зафиксирован перед тем, как эти алгоритмы применялись.) Имеет место тривиальная атака нахождения коллизии, запущенная во время $O(2^A)$: просто определите H на 2^{A+1} различных входящих элемента; по принципу Дирихле два выходных элемента должны быть равны. Можем ли мы сделать еще лучше?

Обобщая вышеописанный алгоритм, скажем, мы выберем q различных входных элементов $x1, \ldots, xq$, вычислим $yi := H(xi)$ и проверим, равны ли какие-либо из двух значений yi . Какова вероятность того, что данный алгоритм найдет коллизию? Как только что было сказано, если $q > 2^A$, тогда коллизия случается с вероятностью 1. Какова вероятность коллизии, если q меньше? Это довольно тяжело анализировать, и поэтому мы лучше проанализируем идеализированный случай, в котором H расценивается как случайная функция. 1 То есть для

каждой i мы предположим, что значение yi = H(xi) равномерно аспределено в {0, 1}A и независимо от любых предыдущих выходных значений {yj }j<i (вспомним, как мы предполагали, что все {xi} различны). Таким образом, мы уменьшили нашу проблему до следующей: если мы выберем значения y1, . . . , yq ∈ {0, 1}A единообразно по случайному принципу, какова будет вероятность, что имеют место различные i, j при yi = yj ?

Данная проблема была внимательно изучена и отнесена к так называемой проблеме «дней рождения», подробно описанной в Дополнении A.4. По этой причине алгоритм нахождения коллизий, который мы описали, часто зовется атакой «дней рождений». Проблема «дней рождений» заключается в следующем: если в комнате находится q людей, какова вероятность того, что у двоих из них день рождения в один и тот же день (Предположим, что дни рождения равномерно распределены по периоду в 365 дней невисокосного года.) Это аналогично нашей проблеме: если yi представляет день рождения человека i, тогда мы имеем y1, . . . , yq ∈ {1, . . . , 365}, выбранные единообразно, и совпадающие дни рождения соответствуют различным i, j при yi = yj (то есть совпадающие дни рождения соответствуют коллизиям).

В Дополнении A.4 мы показываем, что для y1, . . . , yq, выбранных единообразно в {1, . . . , N }, вероятность коллизиии приблизительно 1/2, если $q = \Theta(N^{1/2})$. В случае с дням и рождения, если имеется всего 23 человека, вероятность того, что у двоих из них день рождение выпадет на одну и ту же дату больше, чем 1/2. В нашем условии это означает, что если хэш-функция имеет выходную длину A (и поэтому диапазон будет размером 2^A), то принимая $q = \Theta(2^{A/2})$, выдаст коллизию с вероятностью примерно 1/2.

С точки зрения конкретной защиты вышеописанное означает, что для хэш-функции, чтобы противостоять атакам нахождения коллизии, которые действуют в течение времени T (где мы берем время, чтобы определить H в качестве нашей единицы времени), выходная длина хэш-функции должна быть как минимум 2 log T бит (так как 2(2 log T)/2 = T). Принимая специфические параметры, это означает, что если мы хотим, что нахождение коллизий было настолько же трудным, насколько истощающим является поиск по 128-битным ключам, тогда нам необходимо, чтобы выходная длина хэш-функции была как минимум 256 бит. Мы подчеркиваем, что выходные данные такой длины - это единственное необходимое условие, но не достаточное. Мы также отмечаем, что атаки «дней рождения» работают только для нахождения коллизий. Не существует типичных атак

1Можно увидеть, что это (в значительной степени) наихудший случай, и коллизии случаются с более высокой вероятностью, если H отколняется от случайного, и {xi} выбираются единообразно.

для стойкости второго прообраза или стойкости прообраза хэш-функций H, которые требуют менее чем 2^A оценок H (впрочем см. Раздел 5.4.3).

Нахождение значимых коллизий . Атака «дней рождения», описанная выше, дает коллизию, которая не обязательно очень полезная. Но та же идея может быть использована, чтобы найти и «значимые» коллизии. Предположим, Алиса желает найти два сообщения x и x^r так, чтобы $H(x) = H(x^r)$, и, более того, x должно быть письмом от ее работодателя, объясняющего, почему она была уволена с работы, тогда как xr должно быть преувеличенное рекомендательное письмо. (Это должно позволить Алисе подделать соответствующий тег на рекомендательное письмо, если ее работодатель использует подход хэш-и-MAC для аутентификации сообщений.) Очевидно, что атака «дней рождения» потребует только, чтобы входные данные хэша x1, . . . , xq были различные; им не обязательно быть случайными. Алиса может выполнить атаку по типу «дней рождения» путем генерирования q = Θ(2A/2) сообщение первого типа и q сообщений второго типа и затем поиска коллизий между сообщениями двух типов. Небольшое изменение анализа из Дополнения A.4 показывает, что это дает коллизию между сообщениями различных типов с вероятностью приблизительно 1/2. Немного размышлений, и очевидно, что легко написать одно и то же сообщение многими различными способами. Например, рассмотрите следующее:

> Это тяжело/трудно/затруднительно/невозможно представить/вообразить, что мы найдем/пригласим/наймем другого работника/сотрудника, который обладает теми же качествами/навыками/способностями, что и Алиса. Она сделала прекрасную/превосходную работу.

Любая комбинация слов, выделенных курсивом, возможна и выражает одну и ту же точку зрения. Таким образом, предложение может быть написано 4 • 2 • 3 • 2 • 3 • 2 = 288 различными способами. Это всего лишь одно предложение, и поэтому на самом деле легко сгенерировать сообщение, которое может быть переписано 264 различными способами, все, что нужно - это 64 слова с одим синонимом для каждого. Алиса может приготовить 2A/2 письма, объясняющие, почему она была уволена и еще 2A/2 реко мендательных письма; с высокой вероятностью между двумя типами писем будет найдена коллизия.

5.4.2 Атаки «дней рождения « малого пространства

Атаки «дней рождения», описанные выше, требуют большого количества памяти; они требуют от атакующего хранить все $O(q) = O(2^{A/2})$ значения {yi}, потмоу что атакующий не знает заранее, какая пара значение вызовет коллизию. Это серьезный недостаток, потому что память, в общем, является более ограниченным ресурсом, чем время. И, пожалуй, сложнее разместить и организовать хранилище для 2^{60} байтов, чем выполнить 2^{60} команд процессора.

Более того, всегда можно позволить вычислению работать автономно, в то же время требования алгоритма к памяти должны быть удовлетворены сразу же, как эта память потребуется.

Мы покажем здесь более продвинутую атаку «дней рождения» с существенно уменьшенными требованиями к памяти. В действительности, она имеет такую же временную сложность и вероятность успеха, как и прежняя, однако использует только постоянный объем памяти. Атака начинается с подбора случайного значения x0 и вычисления xi := H(xi−1) и x2i := H(H(x2(i−1))) для i = 1, 2, ... (Обратите внимание, что xi = H(i)(x0) для всех i, где H(i) ссылается на i-кратную итерацию H.) На каждом шаге значения xi и x2i сравниваются; если они идентичны, значит коллизия происходит где-то в последовательности x0, x1, . . . , x2i−1. Затем алгоритм находит наименьшее значение j , для которого xj = xj+i (обратите внимание, что j ≤ i, так как j = i работатет) и выводит xj−1, xj+i−1 в качестве коллизии. Данная атака, описанная формально как Алгоритм 5.9 и проанализированная ниже, требует только хранилища для двух хэш-значений при каждой итерации.

АЛГОРИТМ 5.9
Атака "дней рождения" малого пространства
Вход: Хэш-функция H : $\{0,1\}^* \rightarrow \{0,1\}^A$
Выход: Различные x, x^r при H(x) = H(x^r)

$x_0 \leftarrow \{0,1\}^{\ell+1}$
$x' := x := x_0$
for $i = 1, 2, \ldots$ **do**:
  $x := H(x)$
  $x' := H(H(x'))$
  // now $x = H^{(i)}(x_0)$ and $x' = H^{(2i)}(x_0)$
  if $x = x'$ **break**
$x' := x,\ x := x_0$
for $j = 1$ to i:
  if $H(x) = H(x')$ **return** x, x' and **halt**
  else $x := H(x),\ x' := H(x')$
  // now $x = H^{(j)}(x_0)$ and $x' = H^{(i+j)}(x_0)$

Сколько итераций первого круга мы прогнозировали ранее x^r = x? РАссмотрите последовательность значений x1, x2, . . ., где xi = H(i)(x0), как было определено ранее. Есл и мы смоделируем H в качестве случайной функции, каждое из этих значений будет равномерно и независимо распределяться в $\{0,1\}^A$, пока не случится первый повтор. Таким образом, мы ожидаем, что повтор случится с вероятностью 1/2 в первых q = $\Theta(2^{A/2})$ членах последовательности. Мы покажем, что повтор случится в первых q элементах, алгоритм найдет повтор в максимум q итерациях первого круга:

УТВЕРЖДЕНИЕ 5.10 *Пусть $x_1, \ldots, x_q$ будет последовательностью значений с xm = H(xm−1). Если $x_I = x_J$ при $1 \le I < J \le q$, тогда i < J так, чтобы $x_i = x_{2i}$.*

ДОКАЗАТЕЛЬСТВО Последовательность $x_I, x_{I+1}, \ldots$ повторяется с периодом $\Delta = J - I$. То есть для всех $i \geq I$ и $k \geq 0$ справедливо, что $xi = xi+k \bullet \Delta$. Пусть i будет наименьшим кратным Δ, которое также больше или равно I. Мы имеем $i < J$, так как последовательность значений $\Delta I, I + 1, \ldots I + (\Delta - 1) = J - 1$ содержит кратное Δ. Так как $i \geq I$ и $2i - i = i$ - это кратное Δ, следует, из этого следует, что $xi = x2i$.

Таким образом, если имеет место повторяющееся значение в последовательности $x_1, \ldots, xq$, тогда имеет место $i < q$, для которого $xi = x2i$. Но затем в итерации i нашего алгоритма мы имеем $x = xr$, и алгоритм выходит из первого круга. В этой точке алгоритма мы знаем, что $xi = xi+i$. Затем алгоритм устанавливает $x^r := x(= xi)$ и $x := x_0$, и продолжает искать наименьшее $j \geq 0$, для которого $x_j = x_{j+i}$. (Обратите внимание, что $j \neq 0$, потому что $|x0| = A + 1$.) Он выводит x_{j-1}, x_{j+i-1} в качестве коллизии.

Нахождение значимых коллизий. Алгоритм, описанный выше, Тем не менее, мы покажем, что нахождение значимых коллизий возможно. Трюк в том, чтобы найти коллизию в правильной функции!

Предположим, как и ранее, что Алиса желает найти коллизию между сообщениями двух различных «типов», то есть письмо, объясняющее, почему Алиса была уволена, и преу величенное рекомендательное письмо, которые оба хэшируются до одного и того же значения. Затем Алиса пишет каждое сообщение так, чтобы в них было $A - 1$ взаимозаменяемых слов; то есть имеет место 2^{A-1} сообщений каждого типа. Определите «one-to-one» функцию $g : \{0, 1\}^A \rightarrow \{0, 1\}^*$, так что A входного элемента выбирает между сообщениями типа 0 или типа 1, а бит i (для $1 \leq i \leq A - 1$) выбирает между вариантами взаимозаменяемого слова i в сообщениях соотвествующего типа. Например, рассмотрите предложение:

0: Боб - *хороший/усердный и честный/добросовестный работник/сотрудник*. 1: Боб - *трудный/проблематичный и напрягающий/раздражающий работник/сотрудник*.

Определите функцию g, которая принимает 4-битный входной элемент, где последний бит определяет тип выходного предложения, а начальные три бита определяют выбор слов в предложении. Например:

g(0000) = Боб - хороший и честный работник.

g(0001) = Боб - трудный и напрягающий работник.

g(1010) = Боб - усердный и честный сотрудник.

g(1011) = Боб проблематичный и напрягающий сотрудник.

Теперь определите $f: \{0, 1\}^A \rightarrow \{0, 1\}^A$ по $f(x) \stackrel{\text{def}}{=} H(g(x))$. Алиса может найти коллизию в f, используя атаку «дней рождения» малого пространства, показанную ранее. Суть в том, что любая коллзия x, x^r в f дает два сообщения g(x), $g(x^r)$, которые сталки-

ваются под H. Если x, x^r является случайной коллизией, тогда мы прогнозируем, что с вероятностью 1/2 сталкивающиеся сообщения g(x), g(x^r) будут различных типов (так как x и x^r отличаются последним битом с такой вероятностью). Если сталкивающиеся сообщения одинакового типа, процесс может быть повторен вновь с нуля.

5.4.3 *Компромиссы времени/пространства для обратных функций

В этом разделе мы рассмотрим вопрос устойчивости прообраза, то есть нас интересуют алгоритмы для проблем инверсии функции. Итак, дан алгоритм y = H(x) для универсального x, целью является найти какие-либо x, чтобы H(xr) = y. Начнем, предполагая, что длина входных и выходных данных H одинакова, и вкратце рассмотрим более общий случай в конце.

Пусть h : $\{0, 1\}^A \rightarrow \{0, 1\}^A$ будет функцией. Без эксплуатации каких-либо слабостей H нахождение прообраза точки y может быть осуществлено за время O(2A) посредством изнурительного поиска по области. Мы покажем, что при помощи предварительной обработки и относительно большого объема памяти, это можно сделать эффективнее.

Внесу ясность: мы рассматриваем предварительную обработку как одноразовую операцию и мы не будем сильно зацикливаться на затратах на нее. Нас вместо этого интересует реальное время для инвертирования H в точке y после того, как предварительная обработка уже осуществлена. Это обосновано, если затраты на предварительную обработку могут быть амортизированы через инверсию многих точек, или если у нас есть желание вкладывать в вычислительные ресурсы для предварительной обработки перед тем, как y станет известным для выгоды более быстрого последующего инвертирования. Фактически, это обычное дело - использовать предварительную обработку для обеспечения инверсии функций за очень короткое время. Все, что нам нужно - определить H на каждой точке во время фазы предварительной обработки, а затем сохранить пары {(x, H(x))} в таблицу, отсортированную по второй записи. По достижении любой точки y прообраз y может быть с легкостью найден путем поиска в таблице пары со второй записью y. Недостаток состоит в том, что нам нужно пространство для хранения пар O(2^A) в таблице, которое может быть ограничено, если не недоступно для больших A (то есть A = 80).

Начальная атака методом «грубой силы» использует постоянный объем памяти и время O(2^A), тогда как атака, описанная выше, хранит точки O(2^A) и осуществляет инверсию, по сути, в течение постоянного времени. А теперь мы представим подход, который позволит атакующему сбалансировать время и ппамять. В частности, мы покажем как хранить точки O($2^{2A/3}$) и находить прообраз за время O(22A/3); другие компромиссы также возможны.

Разминка. Hhere the function ачнем с рассмотрения простого случая, где функция H определяет цикл, то есть x, H(x), H(H(x)), . . . охватывает все $\{0, 1\}^A$ для

любой точки отсчета x (обратите внимание, что большинство функций не определяют цикл, но мы предположим такое, чтобы продемонстрировать идею на очень простом случае). Для ясности, пусть $N = 2^A$ обозначает размер области.

В фазе предварительной обработки атакующий просто завершит внутренний цикл, начиная с произвольной точки отсчета x0 и вычисляя $x1 := H(x_0)$, $x2 := H(H(x_0))$ вплоть до $xN = H(N)(x_0)$, где H(i) ссылается i-кратное определение H. Пусть $xi \stackrel{\text{def}}{=} H(i)(x_0)$. Представим разделение цикла на √N сегментов длиной √N каждый, and having the attacker store the points at the beginning и конец каждого такого сегмента. То есть атакующий хранит в таблице пары формы $(xi \bullet \sqrt{N}, x(i+1) \bullet \sqrt{N})$ для i = 0 до √N-1, отсортированные по второму компоненту каждой пары. Итоговая таблица содержит O(√N) точек.

Когда атакующий получает точку y , чтобы инвертировать в онлайн фазу, он проверяет, какая из y, H(y), H(2)(y), . . . соответствует конечной точке сегмента. (Каждая проверка всего лишь задействует поиск по таблице второго компонента хранящейся пары.) Так как y лежит в каком-то сегменте, конечная точка будет гарантировано найдена за √N шагов. После того, как конечная точка $x^r = x\, i\sqrt{N}$ идентифицирована, атакующий берет начальную точку $xr = xi \bullet \sqrt{N}$ соотвествующего сегмента

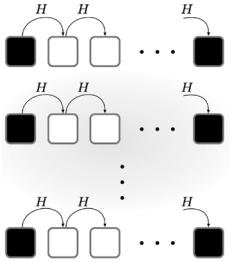

РИСУНОК 5.3: Генерирование таблицы. Хранятся только пары (SPi, EPi).

и вычисляет $H(x^r)$, $H(^2)(x^r)$, . . ., пока не достигается y; что мгновенно дает искомый прообраз. Заметьте, что на это уходит максимум √N определений H. В итоге, такая атака хранит O (√N) точек и находит прообраз с вероятность 1, используя O(√N) хэш-вычислений.

Компромисс времени/пространства Хеллмана. Мартин Хеллман представил более общий компромисс времени/пространства, применимый к произвольной функции H(хотя анализ считает H соучайной функцией). Атака Хеллмана все еще хранит начальную точку и конечную точку нескольких сегментов, но в данном случае сегменты скорее «независимы», чем часть одного большого цикла. Более подробно: пусть s, t будут параметрами, которые мы зададим позже. Атака для начала подбирает s одинаковых начальных точек $SP1, \ldots, SPs \in \{0,1\}^A$. Для каждой такой точки SPi она вычисляетсоответсвующую конечную точку $EPi := H(t)(SPi)$ при помощи t-кратного применения H. (См. Рисунок 5.3.) Затем атакующий хранит значения $\{(SPi, EPi)\}^s$ в

таблице, отсортированной по второй записи каждой пары.

По получении y для инвертирования, атака продолжается так, как в простом случае, описанном ранее. В частности, она проверяет, является ли какое-либо из значений y, $H(y), \ldots, H^{(t-1)}(y)$ равной конечной точке одного из сегментов (останавливаясь как только первое из таких совпадений будет найдено). Возможно такое, что ни одно из этих значений не будет равно конечной точке (что мы обсудим ниже). Однако, если $H(j)(y) = EPi = H^{(t)}(SP_i)$ для некоторых i, j, тогда атакующий вычисляет H(t−j−1)(SPi) и проверяет, является ли это прообразом y. Весь процесс требует максимум t определений H.

Вроде бы, это работает, но есть ряд тонкостей, которые мы игнорируем. Во-первых, может случится, что ни одно из значений y, $H(y), \ldots, H^{(t-1)}(y)$ не является конечной точкой сегмента. Такое может случиться, если y не находится в коллекции значений $s \cdot t$ (не считая начальную точку), полученных во время начального процесса генерирования таблицы. Мы можем установить $s \cdot t \geq N$ в попытке включить каждую A-битную строку в таблицу, но это не решает проблему, так как могут случаться коллизии в самой таблице, фактически, из-за $s \cdot t \geq N$ 1/2 наш предыдущий анализ проблемы «дней рождений» подсказывает нам, что вероятны коллизии, которые уменьшат количество разных точек в коллекции значений. Во-вторых, проблема, которая возникает, даже еслиу присутствует в таблице, заключается в том, что даже если мы найдем совпадающюю конечную точку, и таким образом $H(j)(y) = EPi = H(t)(SPi)$ для некоторых i, j, это не гарантирует, что $H^{(t-j-1)}(SPi)$ - это прообраз y. Проблема в том, что сегмент y, $H(y), \ldots, H^{(t-1)}(y)$ мог бы столкнуться с сегментом i, даже если y сам по себе не находится в этом сегменте; см. Рисунок 5.4. (Даже если y лежит в каком-то сегменте, первая совпадающая конечная точка может находится не в этом сегменте.) Мы называем это ложное срабатывание. Можно подумать, что это маловероятно, если H является стойкой к коллизиям; и нова, однако, мы имеем дело с ситуацией, когда задействовано более $\sqrt{N}$ точек, и поэтому коллизии становятся более вероятным явлением.

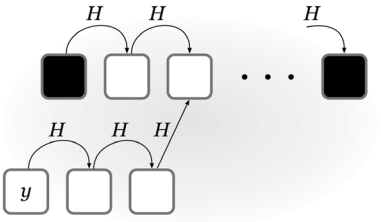

РИСУНОК 5.4: Столкновение в онлайн фазе.

Проблема ложных срабатываний может быть решена путем изменения алгоритма таким образом, чтобы он всегда вычислял полную последовательность

y, H(y), . . . , $H^{(t-1)}$(y) и проверяет, является ли $H^{(t-j-1)}$(SPi) прообразом y для каждого i, j так, чтобы H(j)(y) = EPi. Это гарантирует нахождение прообраза, пока y находится в коллекции значений (не включая начальные точки), сгенерированных во время предварительной обработки. Проблема сейчас в том, что время работы алгоритма может увеличиться, так как каждое ложное срабатывание вызывает дополнительные O(t) хэш-определений. Можно показать, что прогнозируемое количество ложных страбатываний - $O(st^2/N)$. (В последовательности y, H(y), . . . , $H^{(t-1)}$(y) имеются t значений, и в таблице - максимум st разных точек. Рассматривая H как случайную функцию, вероятность того, что какая-либо точка в последовательности равняется какой-либо точке в таблице - 1/N . Прогнозируемое число ложных срабатываний, таким образом, t•st•1/N = st^2/N.) Следовательно, пока $st^2 \approx N$, в чем мы убедимся по другим причинам ниже, прогнозируемое число ложных срабатываний будет постоянным, и обращение с ложными срабатываниями потребует только O(t) дополнительных хэш-вычислений.

Учитывая вышеописанное изменение, вероятность инвертирования y = H(x) - это как минимум вероятность того, что x находится в коллекции точек (не включая конечные точки), сгенерированные во время предварительной обработки. Мы сейчас снижаем границу этой вероятности, захватив произвольность процесса предварительной обработки, равно как и единообразный выбор x, рассматривая H как случайную функцию в анализе. Для начала вычислим прогнозируемое число разных точек в таблице. Рассмотрим, что случится, когда будет сгенерирована строка i таблицы. Начальная точка SPi универсальна и в таблице уже максимум (i − 1) • t разных точек (не включая конечные точки), потому ветроятность того, что SP_i «новая» (то есть не равная любому из предыдущих значений) равняется как минимум 1 − (i − 1) • t/N. Какова вероятность того, что $H(SP_i)$ новая? Если SP_i не новая, то $H(SP_i)$ почта наверное тоже. С другой стороны, если SP_i новая, тогда $H(SP_i)$ является универсальной (потому что мы рассматриваем H как случайную функцию) и поэтому новой с вероятностью как минимум 1 − ((i − 1) • t + 1)/N . (теперь у нас есть дополнительная точка SPi.) Таким образом, вероятность того, что H(SPi) новая - как минимум

$$\Pr[SP_i \text{ is new}] \cdot \Pr[H(SP_i) \text{ is new} \mid SP_i \text{ is new}]$$
$$\geq \left(1 - \frac{(i-1)\cdot t}{N}\right) \cdot \left(1 - \frac{(i-1)\cdot t + 1}{N}\right)$$
$$> \left(1 - \frac{(i-1)\cdot t + 1}{N}\right)^2.$$

Продолжая идти этим путем, вероятность того, что H(t−1)(SPi) новая - как минимум

$$\left(1 - \frac{i\cdot t}{N}\right)^t = \left[\left(1 - \frac{i\cdot t}{N}\right)^{\frac{N}{i\cdot t}}\right]^{\frac{i\cdot t^2}{N}} \approx e^{-it^2/N}.$$

На что тут следует обратить внимание, так это когда $it^2 \leq N/2$, вероятность этого как минимум 1/2; с другой стороны, так как $it2 > N$, вероятность довольно мала. Рассматривая последнюю строку, когда i = s, это ознаает, что мы не получим значительного дополнительного покрытия, если $st^2 > N$. Хорошим условием для параметров, таким образом, будет $st^2 = N/2$. Учитывая это, прогнозируемое число разных точек в таблице

$$\sum_{i=1}^{s}\sum_{j=0}^{t-1} \Pr\left[H^{(j)}(SP_i) \text{ is new}\right] \geq \sum_{i=1}^{s}\sum_{j=0}^{t-1} \frac{1}{2} = \frac{st}{2}.$$

Вероятность того, что x «покрыто» - как минимум $\frac{st}{2N} = \frac{1}{4t}$

Это дает слабый компромисс времени/пространства, в котором мы можем использовать больше пространства (и, соответственно, меньше времени) за счет уменьшающейся вероятности инвертирования y. Но мы можем сделать лучше путем генерирования T = 4t «независимых таблиц» (Это увеличит как пространство, так и время на коэффициент T .) Пока мы можем рассматривать вероятности нахождения x в каждой из связанных таблиц как независимую, вероятность того, что как минимум одна из этих таблиц содержит x -

$$1 - \Pr[\text{no table contains x}] = 1 - \left(1 - \frac{1}{4t}\right)^{4t} \approx 1 - e^{-1} = 0.63.$$

Единственный оставшийся вопрос - это как сгенерировать независимую таблицу. (обратите внимание, что генерирование таблицы точно так же, как и ранее, - это то же самое, что и добавлять s дополнительных строк к нашей первоначальной таблице, что, как мы могли наблюдать, не очень помогает.) Мы можем это сделать для такой таблицы i путем применения какой-то функции Fi после каждого вычисления H, где F1, . . . , FT все разные. (Хорошим выбором могла бы быть установка $Fi(x) = x \oplus ci$ для какой-то фиксированной константы ci , которая разная в каждой таблице.) Пусть Hi $\stackrel{\text{ob}}{=}$ Fi ◦ H, то есть Hi(x) = Fi(H(x)). Затем для таблицы i мы снова подбираем s случайных начальных точек, но для каждой такой точки мы теперь вычисляем Hi(SP), $H^{(2)}$(SP), и так далее. По получении значения y = H(x) для инвертирования, атакующий сначала вычисляет y^r = Fi(y) и затем проверяет, отвечают ли какие-либо из yr, Hi$^{(yr)}$, . . . , $H^{(t-1)}$ (y^r) конечной точке в таблице i; это повторяется для i = 1, . . . , T . (дальнейшие детали мы опускаем.) В то время, как тяжело аргементировать независимость формально, данный подход приводит к хорошим результатам на практике.

Выбирая параметры. Пожытоживая вышеописанно, мы видим, что пока $st^2 = N/2$, мы имеем алгоритм, который хранит O(s • T) = O(s •t) = O(N/t) точек во время фазы предварительной обработки, и можем инвертировать y с постоянной вероятностью за время O(t• T) = O(t^2). Одно условия параметров - это $t = N^{1/3} = 2^{A/3}$, при котором мы имеем алгоритм, сохраняющий O(22A/3) точек, который находит

прообразы с использованием $O(2^{2A/3})$ хэш-вычислений. Если используется хэш-функция с 80 битами выходных данных, тогда это реализуемо на практике.

Обрабатывая разные область и диапазон. Естественным явлением на практике есть попадание в ситуацию, в которой область и диапазон H разные. Один из примеров лежит в контексте взлома паролей (см. Раздел 5.6.3), когда атакующий имеет H(pw), но |pw| A. В общем случае, скажем, x выбран из какой-то области D, которая может быть больше или меньше, чем $\{0, 1\}^A$. И хотя, конечно же, есть возможность искусственно расширить область/диапазон, чтобы их размеры совпадали, это не будет полезным для атаки, описанной выше. Чтобы понять почему, рассмотрим пример с паролем. Чтобы атака прошла успешно мы хотим, чтобы pw находился в какой-то таблице значений, сгенерированных во время предварительной обработки. Если мы сгенерируем каждую строку таблицы простым вычислением H(SP), $H^{(2)}$(SP), . . . для SP ∈ D, тогда ни одно из этих значений (кроме, возможно, самой SP) не будет равняться pw.

Мы можем добиться этого, применив функцию Fi, как и ранее, между каждым вычислением H, хотя сейчас мы выбираем Fi, преобразовывая $\{0, 1\}^A$ в D. Это решает вышеописанную проблемы, так как Fi(H(SP)), $(Fi \circ H)^{(2)}$(SP), . . . теперь все лежат в D.

Применения к атакам на восстановление ключа. Компромиссы времени/пространства производят атаки и на криптографические примитивы, которые не являются хэш-функциями. Одно каноническое применение, по сути, применение изначально рассматриваемое Хеллманом, - это атака на произвольный блочный шифр F, которая ведет к восстановлению ключа. Определите $H(k) \stackrel{def}{=} F_{k(x)}$, где x - это какой-то произвольный, но фиксированный входной элемент, который будет использоваться для построения таблицы. Если атакующий сможет получить F_k (x) для неизвестного ключа k, то ли посредством атаки на основе подобранного открытого текста, то ли посредством подбора x так, чтобы F_k (x) было получено скорее при помощи атаки на основе открытых текстов, тогда инвертированием H атакующий узнает (возможное значение для) k. Обратите внимание, что длина ключа F , возможно, отличается от длины своего блока, но в этом случае мы можем использовать метод, описанный выше, для обработки H с разными областью и диапазоном .

5.5 Модель со случайным оракулом

Существуют несколько примеров конструкций на основе криптографических хэш-функций, которые не могут быть наверняка защищенными только лишь на основании предположения, что хэш-функция является стойкой к коллизиям или нахождению прообраза. (Мы увидим несколько в следующих разделах.) Во многих случаях, оказывается, нет простых и обоснованных предположений касательно хэш-функции, которая бы наверняка обеспечивала защиту конструкции.

При попадании в такую ситуацию, существуют несколько выходом. Один из таких выходов - найти схемы, которые могут наверняка обеспечить защиту на основе какого-либо обоснованного предположения об испольуемой хэш-функции. Это хороший подход, но он оставляет открытым вопрос о том, что же делать до того, пока такие схемы не будут найдены. Также доказуемо защищенные конструкции могут быть значительно менее эффективные, чем другие подходы, защита которых не была доказана. (Это очевидная проблема, с которой мы столкнемся в условиях крптосистем с открытым ключом.)

Другая возможность, конечно же, - это использование существующей криптосистемы, даже если нет другого обоснования ее защиты, кроме, возможно, того факта, что разработчики постарались атаковать ии и потерпели неудачу. Это идет вразрез со всем, что мы сказали о важности научного, современного подхода к криптографии, и должно быть ясно, что это неприемлемо.

Подход, который был очень успешен на практике и который предлагает «золотую середину» между полным научным доказательством защиты с одной стороны и отсутствием каких-либо доказательств - с другой, заключается в том, чтобы представить идеализированную модель для доказательства защиты криптографических схем. Хотя идеализация может не быть точным отражением реальности, мы можем по крайней мере извлечь некоторые меры уверенности в работоспособности проекта схемы из доказательства в рамках идеализированной модели. Если модель является обоснованной, такие доказательства определенно лучше, чем отсутствие доказательств.

Наиболее популярный пример такого подхода - это модель со случайным оракулом, которая рассматривает криптографическую хэш-функцию H в качестве истинно случайной функции. (Мы уже видели пример этому в нашем описании компромиссов времени/пространства, хотя там мы анализировали скорее атаку, чем конструкцию.) Выражаясь более конкретно, модель со случайным оракулом предполагает существование публичной, случайно функции H , которая может быть вычислена только путем «опроса» оракула, своего рода «черного ящика», который возвращает H(x) после получения водного элемента x. (Мы обсудим, как это можно истолковать, в следующем разделе.) Чтобы разграничить вещи, модтель, которую мы использовали до этого (где не присутствовал случайный оракул) часто называют «стандартной моделью».

Никто не утверждает, что случайный оракул существует, хотя были выдвинуты предположения, что случайный оракул мог бы имплементирован на практике с использованием доверительной стороны (то есть какого-то сервера в интернете). Конечно, модель со случайным оракулом обеспечивает формальную методологию, которая может быть использована для разработки и валидации криптографических схем с использованием следующего двухшагового подхода:

1. Во-первых, схема разрабатывается, и доказывается ее защита, на модели со случайным оракулом. То есть мы предполагаем, что среда содержит случайного оракула и конструирует и анализирует криптографическую схему внутри такой модели. Стандартные криптографические предположения такого типа, которые мы видели и ранее, и теперь могут быть использованы в доказательстве.

2. Когда мы хотим имплементировать схему на практике, случайный оракул становится недоступным. Вместо этого, случайный оракул реализовывается посредством криптографической хэш-функции H. (Мы вернемся к этому моменту в конце раздела.) То есть на каждом этапе, когда схема требует, чтобы сторона опращивала оракула на предмет значения H(x), сторона вместо этого самостоятельно вычисляет Hˆ (x).

Надежда в том, что криптографическая хэш-функция, используемая во время второго шага, является «достаточно хорошей» при эмуляции случайного оракула, поэтому доказательство защиты, полученное во время первого шага, будет перенесено на практическую реализацию схемы. Трудность заключается в том, не существует теоретического обоснования такой надежды, и, фактически, имеют место (искусственные) схемы, защита которых может быть доказана на модели со случайным оракулом, но незащищенные независимо от того, как случайный оракул реализован на втором шаге. Более того, неясно (с математической точки зрения или эвристической), что для хэш-функции значит быть «достаточно хорошей» при эмуляции случайного оракула, и также неясно, можно ли вообще достичь такой цели. В частности, никакая конкретная реализация Hˆникогда не сможет вести себя как случайная функция, так как Hˆявляется детерминированной и фиксированной. По этой причине доказательство защиты на модели со случайным оракулом должно рассматриваться как предоставление подтверждения того, что схема не имеет «внутренних недостатков проектирования», и не является научным доказательством того, что любая реальная реализация схемы защищена. Дальнейшее описание того, как истолковать доказательства в модели со случайным оракулом представлено в Разделе 5.5.2.

5.5.1 Более подробно о модели со случайным оракулом

Перед тем, как продолжить, давайте определим, что же собой представляет модель со случайным оракулом. Хорошим способом представить себе модель со случайным оракулом будет следующий: «Оракул» - это просто коробка, которая принимает бинарную строку в качестве входного элемента и возвращает бинарную строку в качестве выходного элемента. Что происходит внутри коробки - неизвестно и непостижимо. Все, доверенные стороны, равно как и злоумышленник, могут взаимодействовать с коробкой, и такое взаимодействие состоит из подачи бинарной строки x в качестве входного жлемента и получения бинарной строки y в качестве выходного; мы назовем это опрашивание оракула по x

и саму строку x назовем запросом , поданным оракулу . Предполагается, что запросы оракулу должны быть частными, поэтому, если какая-то сторона опрашивает оракула по входному элементу x , то никто более не должен знать x или даже знать, что эта сторона опрашивает оракула вообще. В этом есть смысл, потому что обращения к оракулу соответствуют (в практической реализации) местным вычислениям криптографической хэш-функции. Важным свойством этой «коробки» является то, что она последовательна. То есть, если коробка когда-либо выводила y на конкретный входной элемент x, тогда она всегда выводит один и тот же ответ y после получения такого же входного элемента x снова. Это означает, что мы можем рассматривать коробку, как имплементацию проработанной функции H; то есть мы определяем функцию H на основании входных/выходных характеристик коробки. Для удобства мы, таким образом, говорим «опрашивание H» вместо опраивания коробки. Никто не «знает» полной функции H (кроме самой коробки); в лучшем случае, все, что известно, - это значения H по строкам, которые были открыто запрошены до сих пор.

Мы уже обсудили в Главе 3, что значит подбирать случайную функцию H. Мы только напомним здесь, что существует два эквивалентных способа представить себе единообразный подбор H: либо изобразить H, выбранную «одним махом» единообразно из набора всех функций по какой-то определенной области или диапазону, либо представить себе генерирующиеся выходные данные для H «на лету», как будет необходимо. В частности, во втором случае мы можем рассматривать функцию как такую, что была определена таблицей, которая изначально пуста. Когда оракул получает запрос x , он поначалу проверяет, $x = x_i$ для какой-либо пары (x_i, y_i) в таблице; если да - соответствующее значение y_i возвращается.

Иначе универсальная строка $y \in \{0, 1\}^A$ выбирается (для какой-либо специфической A), возвращается ответ y ; и оракул сохраняет (x, y) в таблицу. Вторая точка зрения часто концептуально более легкая обоснования и также более легкая в техническом плане, если H определяется в бесконечной области (то есть $\{0, 1\}^*$).

Когда мы определили псевдослучайные функции в Разделе 3.5.1, мы также рассмотрели алгоритмы с доступом оракула к случайной функции. Во избежание каких-либо заблуждений, мы отмечаем, что использование случайной функции там очень отличается от использования случайной функции здесь. Там случайная функция была использована в качестве способа определения, что это значит для (конкретной) ключевой функции быть псевдослучайной. В модели со случайным оракулом, наоборот, случайная функция сама по себе используется в качестве конструкции и должна быть каким-то образом реализована на практике, если мы хотим конкретной реализации конструкции. Псевдослучайная функция не является случайным оракулом, потому что она псевдослучайна только при секретном ключе. Однако, в модели со случайным оракулом всем

сторонам необходимо иметь возможность вычислять функцию; таким образом, там может и не быть секретного ключа.

Определения и доказателства в модели со случайным оракулом

Определения модели со случайным оракулом немного отличаются от определений в стандартной модели, потому что области вероятности в каждом случае разные. В стандартной модели схема П является защищенной, если для всех ppt злоумышленников А вероятность какого-либо события - ниже определенного порога, когда такая вероятность берется посредством случайных выборов сторон, запускающих П и выборовзлоумышленника А. Предполагая, что доверенные стороны, которые используют П на практике, делают случайный выбор в соответствии со схемой, удовлетворение определения данного типа гарантирует защиту для использования П на практике.

В модели со случайным оракулом, наоборот, схема П может опираться на оракула Н. Как и ранее П является защищенной, если для всех ppt злоумышленников А вероятность какого-либо события - ниже определенного порога, но теперь эта вероятность берется посредством случайного выбора Н, равно как и случайных выборов сторон, запускающих П и выборов злоумышленника А. При использовании П на практике, некоторый (его реализация) Н должен быть фиксированный. К несчастью, защита П не гарантируется для любого специфического выбора Н. Это показывает на одну из причин того, почему тяжело утверждать, что любая конкретная реализация оракула Н детерминированной функцией дает защищенную схему. (Дополнительная техническая трудность состоит в том, что, когда конкретная функция Н является фиксированной, злоумышленнику А более не запрещено опрашивать Н в качестве оракула, но он может увидеть и использовать код Н во время своей атаки.)

Доказательства в модели со случайным оракулом могут эксплуатировать тот факт, что Н выбирается случайным образом, и что единственный способ вычислить Н(х) - в открытую запросить х у Н. Три особых свойства являются особенно полезными; мы вкратце набросаем их здесь и покажем несколько несколько их простых применений ниже и в следующем разделе, но стоит предупредить, что полное понимание скорее всего подождет до тех пор, пока мы не представим формальные доказательства в модели со случайным оракулом в следующих главах.

Первое полезное свойство модели со случайным оракулом - это:

Если х не был запрошен у Н, тогда значение Н(х) является универсальным. На первый взгляд это может напоминать гарантию, обеспеченную псевдослучайным генератором, но на самом деле это сильнее. Если tt является псевдослучайным генератором, тогда tt(x) является псевдослучайным для наблюдателя, предполагающего, что х выбран единообразно случайным образом и является полностью неизвестным наблюдателю. Если Н является случайным оракулом, однако, тогда

H(x) является истинно универсальным для наблюдателя, посколько наблюдатель не запрашивал x. Это истинно, даже если x является известным, или если x не является универсальным, но является трудноугадываемым. Например, если x - это n-битная строка, где первая половина x известна и вторая половина случайна, тогда tt(x) должно быть легко отличить от случайного, а H(x) - нет.)

Оставшиеся два свойства относятся непосредственно к доказательствам посредством редукции в модели со случайным оракулом. (Возможно, полезно будет обратиться к Разделу 3.3.2.) В качестве части сведения случайный оракул, с которым взаимодействует злоумышленник A , должен быть симулирован. То есть: A будет подавать запросы и получать ответы от того, что он будет представлять как оракула, но редукция сама по себе должна теперь отвечать на данные запросы. Оказывается, это дает много возможностей. Прежде всего:

Если A запрашивает x у H, редукция видит этот запрос и узнает x.

Это иногда называется «экстрагируемость». (Это не противоречит факту, упомянутому ранее, что запросы случайному оракулу являются «частными».) Хотя это является истинным в самой модели со случайным оракулом, здесь мы используем A как подпрограммувнутри редукции, которая симулирует случайного оракула для A.) Наконец:

Редукция может устанавливать значение H(x) (то есть ответ на запрос x) до значения своего выбора, поскольку это значение правильно распределено, то есть является универсальным.

Это называется «программируемость». Не существует аналога экстрагируемости или программируемости, как только There is no counterpart to extractability or programmability once H is instantiated with any concrete function .

Простые иллюстрации модели со случайным оракулом

На этом этапе полехзными могут оказаться некоторые примеры. Примеры, приведенные здесь относительно просты и не используют все возможности, которые дает модель со случайным оракулом. Скорее, эти примеры представлены только, чтобы обеспечить плавное введение в модель. В дальнейшем мы предполагаем случайного оракула, преобразующего Ain-битные входные данные Aout-битные выходные данные, где Ain, Aout > n, параметр защиты (поэтому Ain, Aout являютс функциями n).

Случайный оракул как псевдослучайный генератор. Сначала мы покажем, что для Aout > Ain случайный оракул может быть использован в качестве псевдослучайного генератора. (Мы не говорим, что случайный оракул является псевдослучайным генератором, так как случайны оракул не является фиксированной функцией.) Строго говоря, мы утверждаем, что для любого ppt злоумышленника A существует пренебрежимо малая функция negl такая, что

$$\left| \Pr[\mathcal{A}^{H(\cdot)}(y) = 1] - \Pr[\mathcal{A}^{H(\cdot)}(H(x)) = 1] \right| \leq \mathsf{negl}(n),$$

где в первом случае вероятность берется из универсального выбора H, универсального выбора y ∈{0, 1}Aout(n) и произвольности A, и во втором случае вероятность берется из универсального выбора H, универсального выбора x ∈{0, 1}Ain(n) и произвольности A. Мы явно указали, что A имеет доступ оракула к H в каждом из случаев; как только H выбран, A может свободно подавать ему запросы.

Пусть S обозначает набор точек, по которым A опросил H; конечно же, |S| является полиномиальным в n. Заметьте, что во втором случае вероятность того, что x ∈ S - пренебрежимо мала. Это справедливо, если A начинает без информации об x (обратите внимание, что H(x) сама по себе ничего не показывает об x, потому что H является случайно функцией), и потмоу что S - экспоненциально меньше, чем {0, 1}Ain. Кроме того, зависящий от x $f \notin$ S во втором случае входной элемент A в каждом случае является универсальной строкой, которая независима от ответов на запросы от A.

Случайный оркул вкачестве стойкой к коллизиям хэш-функции. Если $A_{out} < A_{in}$, случайный оракул является стойким к коллизиям. То есть вероятность успеха любого ppt злоумышленника A в следующем эксперименте - пренебрежимо мала:

1. Выбрана случайная функция H .

2. A достигает успеха, если он выводит различные x, x^r при H(x) = H(x^r).

Чтобы это увидеть предположим без потери обобщенности, что A выводит только значения x, xr , которые он ранее запрашивал у оракула, и что A не делает один и тот же запрос оракулу дважды. Разрешив запросам оракулу от A быть x_1, . . . , x_q при q = poly(n), становится ясно, что вероятность того, что A достигнет успеха ограничивается вероятностью того, что H(xi) = H(xj) для некоторых i *f*= j. Но эта вероятность ровно такая же, как и вероятность того, что, если мы возьмем q строк y1, . . . , yq ∈ {0, 1}Aout назависимо и единообразно случайным образом, у нас будет yi = yj для некоторых i *f*= j. Это точь в точь проблема «дней рождений», и поэтому, используя результаты Дополнения A.4, мы имеем то, что A достиг успеха с пренебрежимо малой вероятностью O($q^2/2^{Aout}$).

Конструирование а псевдослучайной функции из случайного оракула. Это такеже достаточно просто сконструировать псевдослучайную функцию в модели со случайным оракулом. Предположим, что $A_{in}(n) = 2n$ и $A_{out}(n) = n$ и определим

F (x) $\stackrel{\text{def}}{=}$ H(k||x),

где |k| = |x| = n. В Упражнении 5.11 вас попросили показать, что это псевдослучайная функция, а именно, что для любого полиномиально-временного A вероятность успехаA в следующем эксперименте - не более 1/2 плюс пренебрежимо

малая функция:

1. Функция H и значения $k \in \{0, 1\}^n$ и $b \in \{0, 1\}$ выбираются единообразно.

2. Если b = 0, злоумышленник A получает доступ к оракулу для Fk (•) = H(k»•). Если b = 1, тогда A получает доступ к случайной функции, преобразующей n-битные входные данные в n-битные выходные данные. (Эта случайная функция является независимой от H.)

3. A выводит бит br, и достигает успеха, если $b^r = b$.

На втором шаге, A может получить доступ к H в дополнение к оракулу функции, предоставленному ему экспериментом. (Псевдослучайная функция в модели со случайным оракулом должна быть неотличима от случайной функции, независимой от H.)

Интересный аспект всех вышеописанных утверждений заключается в том, что они не предполагают вычислительных допущений; они справедливы даже для вычислительно неограниченных злоумышленников, пока такие злоумышленники ограничены полиномиально большим количеством запросов оракулу. Этому нет аналогов в реальности, где, как мы убедились, вычислительные допущения необходимы.

5.5.2 Работает ли метод случайного оракула ?

Схемы, разработанные в модели со случайным оракулом, внедряются на практике путем реализации H м некоторыми конкретными функциями. С механиков модели со случайным оракулом позади нас, мы подходим к более фундаментальному вопросу:

Что гарантируют доказательства защиты в модели со случайным оракулом касательно защиты любой реальной реализации?

Вопрос не имеет определенного ответа: ведутся споры внутри криптографического сообщества касательно того, как толковать доказательства в модели со случайным оракулом, и активная область исследования должна определить, что конкретно доказательство защиты в модели с случайным оракулом предполагают по отношению к реальному миру. Мы можем только надеяться, что споры продолжат мотивировать обе стороны.

Возражения против модели со случайным оракулом . Отправной точкой для аргументов против использования случайных оракулов проста: как мы уже отметили, не существует формального или научного обоснования для того, что доказательство защиты для какой-либо схемы П в модели со случайным оракулом скажет нам что-либо о защите П на практике, после того, как случайный оракул H был реализован с какой-либо определенной хэш-фунцкией Hˆ . Это более чем просто теоретическое неудобство. Небольшие размышления показывают, что ни одна конкретная хэш-функция никогда не сможет действовать как «истинный» случайный оракул. Например, в модели со случайным оракулом

значение H(x) является «полностью случайным», если x не был явно запрошен. Аналог должен был бы потребовать, чтобы Hˆ (x) была случайной (или псевдослучайной), если Hˆ не был явно подсчитан по x. Как мы должны толковать то на практике? Даже не очень ясно, что означает «явно вычислить» Hˆ : что если злоумышленник знает какой-то легкий путь для вычисления Hˆ, который не требует запуска реального кода для Hˆ ? Кроме того, Hˆ (x), возможно, не может быть случайным (или даже псевдослучайным), поскольку после того, как злоумышленник узнает описание Hˆ, значение этой функции по всем входным данным будет тут же определено.

Ограничения модели со случайным оракулом становятся более понятными после того, как мы исследуем доказательные методы, представленные ранее. Вспомним, что одним из доказательных методов является использование факта, что редукция может «увидеть» запросы, которые злоумышленник A совершает случайному оракулу. Если мы заменим случайный оракул определенной хэш-функцией Hˆ, это будет означать, что мы должны обеспечить описание Hˆ злоумышленнику в начале эксперимента. Но затем A сможет вычислить Hˆсамостоятельно без выполнения каких-либо явных запросов, и поэтому редукция больше не будет иметь возможность «видеть» запросы, выполняемые A. (Фактически, как было ранее отмечено, понятие о злоумышленнике A, выполняющим явные вычисления Hˆ может не быть истинным и определенно не может быть формально определенным.) Аналогичным образом, доказательства защиты защиты в модели со случаным оракулом позволяют редукции выбирать выходные данные H на свое усмотрение, что-то, что просто невозможно, когда используется конкретная функция.

Даже если мы пожелаем игнорировать, вышеописанные теоретические проблемы, практическая проблема будет в том, что мы на данный момент не имеем полного понимания того, что означает для конкретной хэш-функции быть «достаточно хорошей» при реализации случайного оракула. Для больше конкретики скажем, мы хотим реализовать случайного оракула с использованием некоторой соответствующей модификации HA-1 (SHA-1 - это криптографическая хэш-функция, рассматриваемая в Разделе 6.3.3). Если для какой-то определенной схемы П должно быть обосновано предположить, что П является защищенной при реализации с использованием SHA-1, намного менее обосновано предполагать, что SHA-1 сможет занять место случайного оракула в каждой схеме, разработанной по модели со случайным оракулом. Действительно, как мы уже говорили ранее, мы знаем, что SHA-1 не является случайным оракулом. И не очень-то и сложно разработать схему, которая является защищенной внутри модели со случайным оракулом, но незащищенную, когда SHA-1 заменяет случайного оракула.

Мы настаиваем, что предположение о том, что форма «SHA-1 действует как случайный оракул» качественно отличается от предположения «SHA-1 является стойкой к коллизиям» или «AES является псевдослучайной функцией».

Проблема заключается частично в том факте, что не существует удовлетворительного объяснения тому, что означает первое утверждение в то время, как у нас имеются определения для двух последних утверждений.

Из-за этого использование модели со случайным оракулом для доказательства схемы качественно отличается от, например, представления нового криптографического допущения для того, чтобы доказать, что схема защищена в стандартной модели. Таким образом, доказательства защиты в модели со случайным оракулом являются менее удовлетворительны, чем доказательства защиты в стандартной модели.

Поддержка модели со случайным оракулом . Учитывая все проблемы, связанные с моделью со случайным оракулом, зачем ее вообще использовать? Если говорить более конкретно: почему модель со случайным оракулом стала настолько влиятельна в развитии современной криптографии (и в частности, современного практического использования криптографии), и почему ее продолжают широко использовать? Как мы увидим, модель со случайным оракулом позволяет разрабатывать существенно больше эффективных схем, чем известных нам схем, конструируемых в стандратной модели. В связи с этим, существует всего несколько (если вообще существуют) криптосистем с открытым ключом, используемых сегодня, располагающих доказательствами защиты в модели со случайными оракулами. К тому же, доказательства в модели со случайными оракулами почти повсеместно признаны кредитом доверия защите схем, рассматриваемых для стандартизации.

Фундаментальной причиной тому является убеждение в том, что:

Доказательство защиты в модели со случайным оракулом намного лучше, чем отсутствие доказательства вообще.

Хотя многие не согласятся, мы предлагаем следующее в поддержку данного суждения:

• Доказательство защиты для схемы в модели со случайным оракулом указывает на то, что конструкция схемы «работоспособна» в том смысле, что единственные возможные атаки на практическую реализацию схемы - это те, которые возникают из-за слабости в хэш-функции, использованной для реализации случайного оракула. Таким образом, если «достаточно хорошая» хэш-функция используется для реализации случайного оракула, мы должны иметь уверенность в защите схемы. Кроме того, если данная реализация схемы успешно атакована, мы можем просто заменить используемую хэш-функцию на «более эффективную».

• Важно отметить, что не было еще успешных реальных атак на схемы с доказанной защитой в модели со случайным оракулом, когда случайный оаркул правильно реализован. (Мы сюда не относим атаки на «искусственные» схемы, но отметим, что значительное внимание необходимо уделить реализации модели со случайным оракулом, как показано в Упражнении 5.10.) В этом и

кроется свидетельство полезности модели со случайным оракулом в разработке практических схем.

Тем не менее, вышеописанное, в конечном итоге, представляет только наглядную гипотезу касательно полезности доказательство в модели со случайном оракулом, и, при прочих равных условиях, доказательства без случайного оракула более предпочтительны.

Реализация случайного оракула

Правильно реализованный случайный оракул является деликатной конструкцией, полное его описание выходит далеко за пределы данной книги. Здесь мы только лишь предупреждаем читателя, что использование «готовой» криптографической хэш-функции без модификации - это, в общем-то, не очень эффективный подход. Прежде всего, большинство криптографических хэш-функций сконструированы с использованием парадигмы Меркле-Дамгорда (сравните с Разделом 5.2), которую легко опознать по случайному оракулу, когда допускаются входные данные произвольной длины. (См. Упражнение 5.10.) Также в некоторых конструкциях важно, чтобы выходные данные случайного оракула лежали в определенном диапазоне (например, оракул должен выводить элементы определенной группы), что выливается в сложности.

5.6 Дополнительные применения хэш-функций

Мы завершим данную главу кратким описанием некоторых дополнительных применений криптографических хэш-функций в криптографии и компьютерной безопасности.

5.6.1 Проверка по отпечаткам пальцев и дедубликация

При использовании стойкой к коллизияфм хэш-функции H, хэш (или дайджест) файла служит в качестве уникального идентификатора для этого фала. (Если у другого файла оказался такой же идентификатор, случается коллизия в H). Хэш H(x) файла x напоминает отпечаток пальца, и кто угодно может проверить, одинаковы ли два файла, сравнив их дайджесты. Такая простая идея имеет множество применений.

• *Антивирусная проверка по отпечаткам пальцев:* Антивирусные приложения идентифицируют вирусы и блокируют их либо отправляют в карантин. Один из самых главных шагов по направлению к этой цели - это хранить базу данных, содержащую хэши известных вирусов и затем прогонять скаченное приложение или вложение к электронному письму через эту базу данных. Так как для каждого вируса всего лишь необходимо записать (или/и распространить) короткую строку, задействованные затраты вполне оправданы.

• *Дедубликация:* Дедубликация данных используется для уничтожения копий данных, особенно в контексте облачного хранилища, когда множество пользо-

вателей хранят данные в одном облач ном сервисе. Здесь наблюдается то, что, если множество пользователей желают сохранить один и тот же файл (например, популярное видео), этот файл можно сохранить в од ном экземпляре и не нужно его загружать в облако каждым пользователем. Дедубликация может быть достигнута сначала путем получения хэша фалов пользователей, которые они хотят сохранить; если файл с таким хэшем уже сохранен в облаке, тогда провайдер облачного хранилища может просто добавить указатель к уже существующему файлу, чтобы обозначить, что этот отдельный пользователь также хранит этот файл. Это экономит и связь, и пространство, и работоспособность такой методологии вытекает из стойкости к коллизиям хэш-функции.

• *Общий доступ к файлам peer-to-peer (P2P):* В системе общего доступа к файлам P2P на сервере хранятся таблицы для обеспечения службы поиска файлов. Эти таблицы хранят хэши доступных файлов, которые обеспечивают идентификатор без использования большого объема памяти.

Возможно, это покажется удивительным, что маленький дайджест может уникально идентифицировать каждый файл в мире. Но это гарантия, обеспеченная стойкими к коллизиям хэш-функциями, которая делает их полезными в описанных выше условиях.

5.6.2 Деревья Меркле

Возьмите клиента, который загружает файл x на сервер. Когда клиент позже извлекает x, он хочет быть уверен, что сервер вернет ему оригинальный, неизмененный файл. Клиент мог бы просто сохранить x и проверить, чтобы извлеченный файл был равным x, но это в первую очередь разрушает саму цель использования серве ра. Мы занимаемся поиском решений, при которых хранилище клиента будет небольшим.

Естественное решение - это использовать подход «проверки отпечатков пальцев», описанный выше. Клиент может на локальном ресурсе хранить короткий дайджест h := H(x); когда сервер возвращает искомый файл x^r , клиенту необходимо только проверить, чтобы $H(x^r)$? h.

Что случается, если мы хотим расширить данное решение до множества файлов x1, . . . , xt? Существует два способа сделать это. Один из способов - это просто хэшировать каждый файл отдельно; клиент на локальном ресурсе хранит дайджесты h1, . . . , ht и проверяет извлеченные файлы как и раньше. Недостаток этого способа в том, что хранилище клиента растет линейным образом в t. Еще одна возможность - это хешировать все файлы вместе. То есть клиент может вычслить h := H(x1, . . . , xt) и сохранить только h. Теперь недостаток заключается в том, что, когда клиент хочет извлечь и проверить правильность файла (i) (xi), ему необходимо извлечь все файлы, чтобы рассчитать дайджест.

Деревья Меркле, представленные Ральфом Меркле, предоставляют компромисс между этими двумя крайностями. Дерево Меркле высчитанное по введен-

ным значениям x1, . . . , xt - это просто бинарное дерево глубиной log t, в котором входные данные расположены на листах, и значение каждого внутреннего узла - это хэш значений ее двух дочерних записей; см. Рисунок 5.5. (Мы предполагаем, что t - это степень числа 2; если нет - мы можем исправить некоторые входные значения на null или использовать незаконченное бинарное дерево в зависимости от применения.)

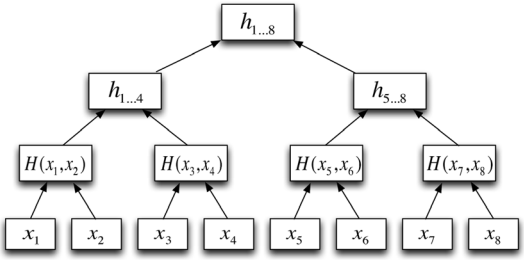

РИСУНОК 5.5 : Дерево Меркле.

Фиксируя какую-то хэш-функцию H, мы обозначаем как MT t функцию, которая принимает t входные значения x1, . . . , xt, вычисляет в итоге дерево Меркле и выводит значение корня дерева. (Ключевая хэш-функция дает ключевую функцию MT t очевидным способом.) Мы имеем:

ТЕОРЕМА 5.11 *Пусть (GenH, H) будет стойкой к коллизиям. Тогда (GenH, MT t) является также стойкой к коллизиям для любых фиксированных t.*

Дерво Меркле таким образом обеспечивает альтернативу преобразованию Меркле-Дамгорда для достижения расширения области для стойких к коллизиям хэш-функций. (Как было сказано, однако, деревья Меркле не являются стойкими к коллизии, если число входных значений t может варьироваться.)

Деревья Меркле обеспечивают эффективное решение нашей первоначальной проблемы, поскольку они позволяют изменять любое оригинальные входные данные t с использованием связи O(log t). Клиент вычисляет h := MT t(x_1, . . . , x_t), загружает x_1, . . . , x_t на сервер и хранит h (вместе с числом файлов t) на локальном ресурсе. Когда клиент извлекает файл i, сервер посылает xi вместе с «доказательством»πi того, что это правильное значение. Это доказательство состоит из значений узлов на дереве Меркле, которые расположены рядом с путем из xi в корень. Посредством этих значений клиент может пересчитать значение корня и проверить, что это значение было равным сохраненному значению h. В качестве примера рассмотрим дерево Меркле на Рисунке 5.5. Клиент вычисляет h1...8 := MT 8(x_1, . . . , x_8), загружает x_1, . . . , x_8 на сервер и ъранит $h_{1...8}$ на локальном ресурсе. Когда клиент извлекает x_3, сервер отправляет x_3 вместе с x_4, $h_{1...2} = H(x_1, x_2)$ и $h_{5...8} = H(H(x_5, x_6), H(x_7, x_8))$. (Если файлы большого размера, мы можем пожелать избежать отправки

любого другого файла, кроме того, который был запрошен клиентом. Это можно с легкостью сделать, если мы определим дерево Меркле по хэшам файлов, а не по самим файлам. Детали опустим.) Клиент вычисляет $h^r := H(h_{1...2}, H(x_3, x_4))$ и $h^r ?r1...4, h_{5...8})$ и затем проверяет, чтобы $h_{1...8} = h^r$

Если H является стойкой к коллизиям, для сервера невозможно отправлять неправильные файлы (и любое доказательство), которые пройдут успешную верификацию. Используя данный подход, локальное хранилище клиента является постоянным (независимо от количества файлов t), и связь от сервера к клиенту пропорциональна log t.

5.6.3 Хэширование паролей

Одним из наиболее распространенных и важных применений хэш-функций в компьютерной безопасности является защита паролей. Рассмотрим пользователя, который вводит пароль перед началом работы на ноутбуке. Чтобы аутентифицировать пользователя какая-то форма пользовательского пароля должна храниться где-то в ноутбуке. Если пароль пользователя хранится в явном виде, тогда злоумышленник, который украдет ноутбук, сможет прочитать пароль пользователя с жесткого диска и войти в ноутбук в роли пользователя. (Это может показаться бессмысленным - прятать пароль от атакующего, который уже может получить доступ к содержимому жесткого диска. Однако, файлы на жестком диске могут быть зашифрованы ключом, извлекаемым из пользовательского пароля, и, таким образом, и доступ к ним может открыться только после ввода пароля. К тому же, пользователи часто используют один и тот же пароль в других местах.)

Такой риск может быть нивелирован посредством хранения хэша пароля вместо самого пароля. То есть жесткий диск хранит значение hpw = H(pw) в файле паролей; когда пользователь вводит свой пароль pw, операционная система проверяет, H(pw) =? hpw перед тем, как открыть доступ. Такой же базовый подход также используется для аутентификации на основе паролей в интернете. Теперь, если атакующий украдет жесткий диск (или взломает веб-сервер), все, что он получит - это хэш пароля, но не сам пароль.

Если пароль подбирается из какого-то относительно небольшого пространства D возможностей (например, D могло бы быть словарем английских слов, в этом случае $|D| \approx 80,000$), атакующий может перебрать все возможные пароли pw1, pw2, . . . $\in$ D и для каждого возможного пароля pwi проверить H(pwi) = hpw. Мы бы хотели заявить, что ничего лучшего атакующий сделать не может. (Это также гарантирует, что злоумышленник не сможет узнать пароль любого другого пользователя, который выбирает сильный пароль из большого пространства.) К несчастью, стойкости прообраза (то есть однонаправленность) H недостаточно, чтобы реализовать то, что мы хотим. Прежде всего, стойкость прообраза только подразумевает, что H(x) тяже-

ло инвертировать, когда x выбирается единообразно из большой области, такой как $\{0, 1\}^n$. Речь не идет о сложности инвертирования H , если x выбирается из како-то другой области, или если x выбирается согласно какому-то другому распределению. Кроме того, стойкость прообраза не подразумевает конкретное количество времени, необходимое для поиска прообраза. Например, хэш-функция

H , для которой вычисление $x \in \{0, 1\}^n$ с учетом H(x) требует время $2^{n/2}$, могла бы все еще считаться стойкой к нахождению прообраза, но это бы все еще значило, что 30-битный пароль мог бы быть восстановлен всего лишь за время 2^{15}.

Если мы смод елируем H как случайный оракул, тогда мы сможем формально доказать защиту, которую мы хотим, а именно, восстановление pw из hpw (предполагая, что pw выбирается единообразно изD) требует |D|/2 вычислений H в среднем.

Вышеописанное подразумевает, что атакующий не осуществлял никакой предварительной обработки. Как мы видели в Разделе 5.4.3, хотя предварительная обработка может быть использована для генерирования больших таблиц, которые обеспечивают инверсию (даже случайной функции!) быстрее, чем метод перебора. Это серьезная проблема на практике: даже если пользователь выбирает пароль как случайную комбинацию из 8 буквенно-цифровых символов, предоставляя парол ю пространство размером $N = 62^8 \approx 2^{47.6}$, может случиться атака с использованием времени и пространства $N^{2/3} \approx 2^{32}$, которая будет очень эффективной. Таблицы могут быть сгенерированы один раз и могут быть использованы для взлома сотен тысяч паролей в случае нарушения работы сервера. Такие атаки рег улярно происходят на практике.

Ликвидация. Мы кратко опишем два механизма, используемых для ликвидации угрозы взлома пароля; дальнейшее описание можно найти в текстах по компьютерной безопасности. Один из методов - это использование «медленных» хэш-функций или замедление существующих хэш-функций с использованием множественных итераций (то есть вычисляя H(I)(pw) для I 1). За этим следует эффект з амедления законных пользователей на коэффициентI, что не является проблемой, если I установлен на какое-то «умеренное» значение (например, 1,000).

С другой стороны, это значительно влияет на злоумышленника, пытающегося взломать тысячи паролей одновременно.

Второй механизм - это ввести соль. Когда пользователь регистрирует свой пароль, ноутбук/сервер генерирует длинное случайное значение s («соль»), уникальное для такого пользователя, и сохраняет (s, hpw = H(s, pw)) вместо просто хранения H(pw) как раньше. Поскольку значение s неизвестно атакующему заранее, предварительная подготовка становится неэффективной, и лучшее, что атакующий может сделать - ждать, пока он получит файл паролей и затем произведет линейный полный перебор области D, как было описано ранее. Об-

ратите внимание, что поскольку разная соль используется для каждого сохраненного пароля, отдельная атака на основе «грубой силы» необходима, чтобы восстановить каждый из паролей.

5.6.4 Установление ключа

Все симметричные криптосистемы, которые мы видели, требуют единообразно распространенную битовую строку для секретного ключа. Часто, однако, боле удобно для дух сторон опираться на общую информацию, например, пароль или биометрические данные, которые не распределяются единообразно. (Забегая вперед, в Главе 10 мы увидим, как стороны могут взаимодействовать для генерирования секрета с высокой энтропией, который не распределяется единообразно.) Стороны могли бы попробовать использовать их общую информацию непосредственно в качестве секретного ключа, но, в общем, это не будет безопасно (поскольку, например, схемы с закрытым ключом все подразумевают единообразно распределенный ключ). Более того, общие данные могут даже не быть корректного формата, чтобы использоваться в качестве секретного ключа (например, они могут быть слишком длинными).

Обрезая общий секрет или преобразовывая его каким-то другим подходящим образом в строку правильной длины, можно потерять значительный объем энтропии. (Мы определили одно понятие об энтропии ниже, но в данный момент следует понимать под энтропией логарифм пространства возможных общих секретов.) Например, представьте две стороны, которые разделяют пароль, состоящий из 28 случайных заглавных букв и хотят использовать криптосистему с 128-битным ключом. Поскольку существует 26 сочетаний для каждого символа, $26^{28} > 21^{30}$ возможных паролей. Если пароль в формате ASCII, каждый символ храниться в 8 битах, и поэтому общая длина пароля будет 224 бита. Если стороны обрезают свой пароль до первых 128 бит, они будут использовать только 16 символов пароля. Однако, это не будет единообрано распределенной 128-битной строкой! Фактически, представления ASCII из букв A–Z лежат между 0x41 и 0x5A; в частности, первые 3 бита каждого байта всегда 010. Это означает, что 37,5% битов итогового ключа будут зафиксированными, и 128-битный ключ, который извлекут стороны, будет иметь только около 75 бит энтропии (то есть будет только 2^{75} или около того сочетаний для ключа.

Что нам требуется, так это типичное решение для извлечения ключа из общего секрета высокой энтропии (но необязательно универсального). Перед продолжением, мы определим понятие энтропии, которое мы здесь рассматриваем.

ОПРЕДЕЛЕНИЕ 5.12 *Распределение вероятностей X имеет m бит минимальной энтропии, m если для каждого фиксированного значения x справедливо, что* $Pr_{X \leftarrow X}[X = x] \leq 2^{-m}$ *–наиболее возможный исход случается с вероятностью максимум* 2^{-m}.

Единообразное распределение по набору размером S имеет минимальную энтропию log S. Распределение, в котором один элемент случается с вероятностью

1/10 и 90 элементов случаются с вероятностью 1/100 имеет минимальную энтропию log 10 ≈ 3.3. Минимальная энтропия распределения измеряет вероятность, с которой атакующий может угадать значение, взятое из распределения; наилучшая стратегия атакующего - угадать наиболее возможное значение, и поэтому, если распределение имеет минимальную энтропию m, атакующий угадает правильно с вероятностью максимум 2^{-m}. Это объясняет, почему минимальная энтропия (в отличие от других понятий энтропии) является полезной в нашем контексте. Расширение минимальной энтропии под названием вычислительная минимальная энтропия определяется, как описано выше, кроме того, что от распределения требуется быть только вычислительно неотличимым от распределения с данной минимальной энтропией. (Понятие вычислительно неотличимости формально определено в Разделе 7.8.)

Функция установления ключа обеспечивает способ получить единообразно распределенную строку из любого распределения с высокой (вычислительной) минимальной энтропией. Не трудно заметить, что, если мы смоделируем хэш-функцию H как случайного оракула, тогда H служит в качестве хорошей функции установления ключа. Рассмотрим неопределенность атакующего касательно H(X), где X взят из распределения с минимальной энтропией m (с технической точки зрения нам требуется, чтобы распределение было независимым от H). Каждый запрос атакующего к H может считаться поп ыткой «угадывания» значения X; по предположению о минимальной энтропии распределения атакующий, делающий q запросов H будет опрашивать H(X) с вероятностью максимум $q \cdot 2^{-m}$. Если атакующий не делает запросов по X к H, тогда H(X) является универсальной строкой.

Также возможно разработать функции установления ключа без необходимости опираться на модель со случайным оракулом с использованием ключевых хэш-функций под названием (сильные) экстракторами. Ключ для экстрактора должен быть универсальным, но необзязательно секретным. Один из стандартов для этого - это HKDF; см. ссылку в конце главы.

5.6.5 Схемы обязательства

Схемы обязательства позволяют одной стороне «привязываться» к сообщению m путем отправления значения обязательства com, при этом получая следующие на первый взгляд противоречивые свойства:

• *Сокрытие:* обязательство о m ничего не отображает.

• *Привязка:* это невозможно, чтобы исполнитель выводил обязательство com, которое он может позже «открыть» как два разных сообщения m, m^r. (В этом смысле com истинно «привязывает» исполнителя к какому-либо четко определенному значению.)

Схема обязательства может рассматриваться как цифровой конверт: запечатывая сообщение в конверт и передавая его другой стороне, можно обеспечить при-

ватность (пока конверт не будет вскрыт) и привязку (пока конверт запечатан).

Строго говоря, (неинтерактивная) схема обязательства определяется случайным алгоритмом Gen, который выводит публичные параметры params, и алгоритма Com , который принимает params и сообщение $m \in \{0, 1\}^n$ и выводит обязательство com; мы сделаем произвольность, используемую Com , явной и обозначим ее как r. Отправитель привязывается к m путем выбора универсальной r, вычисляя com := Com(params, m; r) и отправляя это получателю. Отправитель может позже отвязать com и вскрыть m путем отправления m, r получателю; получатель проверяет это таким образом: Com(params, m; r) =? com.

Сокрытие неформально означает, что com ничего не открывает о m; привязка означает, что невозможно вывести обязательство com , которое может быть открыто двумя различными способами. Сейчас мы формально определим эти свойства.

Эксперимент сокрытия обязательства Hiding $_{A,com}(n)$:

1. Параметры params ← Gen(1^n) генерируются.

2. Злоумышленник A получает входные params и выводит пару сообщений m0, m1 ∈ {0, 1}^{n}.

3. Универсальное значение b∈{0, 1} выбирается и com ← Com(params, mb; r) вычисляется.

4. Злоумышленник A получает com и выводит бит br.

5. Результат эксперимента - 1, если и только если br = b.

Эксперимент привязки обязательства Binding $_{ACom}$ (n):

1. Параметры params ← Gen(1^n) генерируются.

2. Аполучает входные params и выводит (com, m, r, m^r, r^r).

3. Результат эксперимента - определнн как 1, если и только если m f= mr и Com(params, m; r) = com = Com(params, m^r; r^r).

DEFINITION 5.13 *Схема обязательства Com является защищенной, если для всех PPT злоумышленников A существует пренебрежимо малая функция negl такая, чтобы*

$$\Pr\left[\mathsf{Hiding}_{\mathcal{A},\mathsf{Com}}(n) = 1\right] \leq \tfrac{1}{2} + \mathsf{negl}(n)$$

and

$$\Pr\left[\mathsf{Binding}_{\mathcal{A},\mathsf{Com}}(n) = 1\right] \leq \mathsf{negl}(n).$$

Легко сконструировать защищенную схему обязательства из случайного оракула H. Чтобы привязаться к сообщению m, отправитель выбирает универслальное значение $r \in \{0, 1\}^n$ и выводит com := H(m||r). (В модели со случайным оракулом Gen и params не нужны, так как H по сути служит как публичные параметры схемы.) Наглядно, сокрытие проистекает из факта, что злоумыш-

ленник опрашивает H(>||r) с только лишь пренебрежимо малой вероятностью (так как r является универсальной n-битной строкой); если он ни разу не сделает запрос этой формы, тогда H(m»r) ничего не откроет о m. Привязка проистекает из факта, что H является стойкой к коллизиям.

Схемы обязательств могут быть сконструированы без случайных оракулов (фактически, из однонаправленной функции), но подробности находятся за пределами данной книги.

Ссылки и дополнительная литература

Стойкие к коллизиям хэш-функции были формально определены Дамгордом [52]. Дополнительное описание касательно понятий о безопасности для хэш-функций помимо стойкости к коллизиям можно найти в [120, 150]. П реобразование Меркле-Дамгорда было представлено отдельно Дамгордом и Меркле [53, 123]

HMAC был представлен Белларом и др. [14], а позже и стандартизирован [131]. Атаки «дней рождения» малого пространства, описанные в Разделе 5.4.2, опираются на

алгоритм нахождения цикла Флойда (Floyd). Соответстующие алгоритмы описаны по сслыке: http://en.wikipedia.org/wiki/Cycle_detection. Идея нахождения значимых коллизий с использований атак малого пространства по Ювалю (Yuval) [180]. Ключ

параллелизации атак нахождения коллизий, которые могут предложить значительные ускорения на практике, описана в [170]. Компромиссы пространства/времени для функции инверсии были представлены Хеллманом (Helman) [87] с практическими улучшениями, не описанными здесь, предоставленными Ривестом (Risvest) (неопубликовано) и Oechslin [134].

Первая формальная трактовка модели со случайным оракулом была дана Белларом и Рогавэем [21], хотя идея использования «произвольно ищущей» функции в криптографических приложениях была предложена ранее в значительной степени Фиатом (Fiat) и Шамиром (Shamir) [65]. Правильная реализация случайного оракула на основе конкретной криптографической хэш-функции описана в [21, 22, 23, 48]. Фундаментальный негативный результат касательно модели со случайным оракулом принадлежит Канетти (Canetti) и др. [41], который показал (изобрел) схемы, которые защищены в модели со случайным оракулом, но незащищены для любых конкретных реализаций случайного оракула.

Деревья Меркле были представлены в [121]. Установление ключа, используемое на практике, включая HKDF, PBKDF2 и bcrypt. См. [109] для формальной трактовки проблемы и анализа HKDF.

Упражнения

5.1 Представьте формальные определения для стойкости к нахождению второго прообраза и стойкости нахождению прообраза. Докажите, что любая хэш-функция, которая является стойкой к коллизиям, является также стойкой к нахождению второго прообраза, а любая хэш-функция, которая стойкая к нахождению второго прообраза, является также стойкой к нахождению прообраза.

Пусть (Gen_1, H_1) и (Gen_2, H_2) будут двумя хэш-функциями. Определите (Gen, H) так, чтобы Gen запускал Gen_1 и Gen_2 , чтобы получить ключи s_1 и s_2, соответственно. Затем определите $H^{s1,s2}(x) = H^{s1}(x)\|H^{s2}(x)$.

(a) Докажите, что, если по крайней мере одна из (Gen_1, H_1) и (Gen_2, H_2) является стойкой к коллизиям, тогда (Gen, H) является стойкой к коллизиям.

(b) Определите, справедливо ли аналогичное заявление касательно стойкости к нахождению второго прообраза и стойкости к нахождению прообраза, соответственно. Докажите ваш ответ по каждому из случаев.

Пусть (Gen, H) будет стойкой к коллизиям хэш-функцией. Обязательно ли (Gen, Hˆ), определенная посредством $H^{\hat{}s}(x) \stackrel{def}{=} H^s(H^s(x))$, является стойкой к коллизиям?

Предоставьте формальное доказательство Теоремы 5.4 (то есть опишите редукцию).

Обобщите преобразование Меркле-Дамгорда (Конструкция 5.3) для случая, когда хэш-функция фиксированной длины h имеет входную длину n + к (при к > 0) и выходную длину n, и длина входных данных в H должна быть зашифрована как A-битное значение (как описано в Разделе 5.3.2). Докажите стойкойсть к коллизиям (Gen, H), принимая во внимание стойкость к коллизиям (Gen, h).

Для каждой из следующих модификаций преобразования Меркле-Дамгорда (Конструкция 5.3) определите, стойкий ли результат к коллизиям. Если да - предоставьте доказательство; если нет - продемонстрируйте атаку.

(a) Измените конструкцию таким образом, чтобы входная длина не была включена вообще (то есть выведите zB , а не $z_{B+1} = hs(zB\|L)$). (Предположите, что искомая хэш-функция определена только для входных данных, длина которых - это целое число кратное длине блока.)

(b) Измените конструкцию так, чтобы вместо вывода z = hs(zB «L), алгоритм выводил zB ||L.

(c) Вместо использования IV просто начинте вычисление с x1. То есть определите z1 := x1 , а затем вычислите zi := hs(zi 1»xi) для i = 2, . . . , B + 1 и выведите zB+1 , как ранее.

(d) Вместо использования фиксированного IV , установите z0 := L , а затем вычислите zi := h^s(zi 1||x) для i = 1, . . . , B и выведите z .

Предположите, что стойкая к коллизиям хэш-функция имеет место. Покажи-

те конструкцию хэш-функции фиксированной длины (Gen, h), которая не является стойкой к коллизиям, но так, чтобы хэш-функция (Gen, H), полученная из преобразования Меркле-Дамгорда в (Gen, h), как в Конструкции 5.3, была стойкой к коллизиям.

Докажите или опровергните: если (Gen, h) является стойкой к нахождению прообраза, то и хэш-функция (Gen, H), полученная путем применения преобразования Меркле-Дамгорда к (Gen, h), как в Конструкции 5.3, также является стойкой к нахождению прообраза.

Докажите или опровергните: если (Gen, h) является стойкой к нахождению второго прообраза, то и хэш-функция (Gen, H), полученная путем применения преобразования Меркле-Дамгорда к (Gen, h), как в Конструкции 5.3, также является стойкой к нахождению второго прообраза.

Перед HMAC было нормально определять MAC для сообщений произвольной блины путем Macs,k (m) = Hs(k»m), где H - это стойкая к коллизиям хэш-функция.

(a) Покажите, что это совсем не защищенный MAC, если H сконструирована посредством преобразования Меркле-Дамгорда. (Предположите, что хэш-ключ s известен атакующему, и только k содержится в секрете.)

(b) Докажите, что это защищенный MAC, если H смоделирована как случайный оракул.

Докажите, что конструкция псевдослучайной функции, приведенная в Разделе 5.5.1, является защищенной в модели со случайным оракулом.

Докажите Теорему 5.11.

Покажите, как найти коллизию в конструкции дерева Меркле, если значение t нефиксированное. В частности, покажите, как найти два набора входных данных $x_1, \ldots, x_t$ и 1, . . . , x2t так, чтобы MT t(x1, . . . , xt) = MT 2t(x1, . . . , x2t).

Рассмотрите сценарий, представленный в Разделе 5.6.2, в котором клиент хранит файлы на сервере и хочет проверить, чтобы файлы вернулись неизмененными.

(a) Предоствьте формальное определение безопасности для этих условий.

(b) Формализуйте конструкцию, основанную на деревьях Меркле, как описано в Разделе 5.6.2.

(c) Докажите, что конструкция защищенная относительно вашему определению при допущении, что (GenH, H) является стойкой к коллизиям.

Докажите, что схема обязательства, описанная в Разделе 5.6.5, является защищенной в модели со случайным оракулом.

Глава 6

Практические конструкции примитивов с симметричным ключом

В предыдущей главе мы продемонстрировали, как защищенные схемы шифрования и коды аутентификации сообщений могут быть сконструированы из криптографических примитивов, таких как псевдослучайные генераторы, псевдослучайные перестановки и хэш-функции. Один из вопросов, однако, которого мы не коснулись, - это как прежде всего сконструированы эти криптографические примитивы, или существуют ли они вообще! В следующей главе мы изучим этот вопрос с теоретически важной точки зрения и покажем конструкции псевдослучайных генераторов и псевдослучайных перестановок на основе довольно слабых допущений. (Оказывается, хэш-функции намного более сложные конструкции и требуют более сильных допущений. Мы рассмотрим, возможно, защищенную конструкцию хэш-функции в Разделе 8.4.2) В этой главе наше внимание будет приковано к сравнительно эвристическим, но намного более эффективным конструкциям примитивов, которые широко используются на практике.

Как только что было сказано, конструкции, которые мы изучим в этой главе, являются эвристическими в том смысле, что их защищенность нельзя доказать, основываясь на каком-либо слабом допущении. Конструкции, однако, основаны на ряде проектных принципов, некоторые из которых могут быть обоснованы теоретическим анализом. Что, возможно, более важно, многие из этих конструкций пережили годы публичных тщательных проверок и попыток криптоанализа, и, учитывая это, логично было бы предположить, что эти конструкции все же защищены.

В некотором смысле не существует фундаментальных различий между допущением, что, скажем, разложение - это тяжело, и допущением, что AES (блочный шифр, который мы изучим подробно далее в этой главе) является псевдослучайной перестановкой. Однако, существует качественное различие между двумя этими допущениями. 1 Основное отличие в том, что первое допущение более правдоподобное ,так как оно, на первый взгляд, относится к более слабому требованию: допущение, что большие целые числа трудно разложить аргу

1Должно быть ясно, что описание в этом пункте является неформальным, так как мы не можем формально аргументировать что-либо из этого, если мы не можем даже доказать, что разложение - это тяжело!

ментировать намного естественнее, чем допущение, что AES с универсальным ключом неотличим от случайно перестановки. Другие соответствующие различия между допущениями в том, что разложение изучалось намного дольше, чем проблема отличи AES от случайной перстановки, и было признано тяжелой проблемой задолго до появления криптографических схем на его основании.

Подытожим, что логично предположить, что рекомендуемые конструкции, описанные в этой главе защищены, и люди с уверенностью опираются на такие предположения на практике. Все же, предпочтительнее было бы возложить защиту криптографически примитивов на более слабые и более устоявшиеся предположения. Как мы видим в Главе 7, это (в принципе) возможно; к несчастью, конструкции, которые мы увидим там на порядок менее эффективные, чем конструкции, описанные здесь, и сами по себе не очень полезные на практике.

Цель данной главы

Главной целью данной главы является (1) представить некоторые проектные принципы, использованные в конструкции современных криптографических примитивов, и (2) представить читателю некоторые популярные конструкции, широко используемые на практике. Предупреждаем:

• Целью данной главы не является обучение читателей проектированию новых криптографических примитивов. С другой стороны, мы убеждены, что проектирование новых примитивов требует серьезных навыков и усилий и не является чем-то обыденным и легким. Тем, кто заинтересован в развитии дополнительных навыков в этой области, рекомендуем почитать более продвинутую литературу, перечисленную в конце главы.

• Мы не хотим представить все низкоуровневые детали различных примитивов, которые мы здесь опишем, и на наши описания не стоит опираться при имплементации. Фактически, наши описания как бы намеренно неточны, так как мы опускаем некоторые детали, которые не относятся к более широкой концептуальной точке зрения, на которой мы пытаемся акцентировать внимание.

6.1 Потоковые шифры

Вспомним из Раздела 3.3.1, что потоковый шифр определяется двумя детерминированными алгоритмами (Init, GetBits). Алгоритм Init принимает в качестве входных данных ключ k и (опционально) вектор инициализации, IV, и возвращает какое-то начальное состояние st. Алгоритм GetBits может быть использован для генерирования бесконечного потока битов $y_1, y_2, \ldots$ на основе st. Главное требование поточного шифра - чтобы он вел себя как псевдослучайный генератор, а именно, когда k выбирается единообразно случайным образом, итоговая последовательность$y1, y2, \ldots$ должна быть неотличима от последовательности единообразных и независимых битов вычислительно ограниченного атакующего. (В Разделе 3.6.1 мы обращали

внимание на то, что поточные шифры должны иногда отвечать более сильным требованиям безопасности. Мы явно не решаем эту проблему здесь.)

Мы уже выделили (см. конец Раздела 3.5.1), что потоковые шифры могут быть сконструированы из блочных шифров, которые являются более сильными примитивами. Первоначальная мотивация для использования профильных конструкций потоковых шифров, представленных в данном разделе, - это эффективность, особенно в ресурсоограниченных средах (например, в аппаратных средствах, где, возможно, необходимо, чтобы число шлюзов было небольшим). Атаки, проведенные против нескольких свежих конструкций поточных шифров, показали, что их защита, оказывается, намного более слабая, чем в случаях с блочными шифрами. Мы, таким образом, рекомендуем использовать блочные шифры (возможно, в режиме поточных шифров) там где это возможно.

6.1.1 Регистры сдвига с линейной обратной связью

Мы начнем описание регистров сдвига в линейной обратной связью (LFSR). Они использовались исторически для генерирования псевдослучайных чисел, так как они сверхэффективны для внедрения в оборудование и генерируют результат с хорошими статистическими показателями. Однако самостоятельно они не дают криптографически сильные псевдослучайные генераторы и, фактически, мы покажем простую атаку на восстановление ключа на LFSR. Тем не менее, LFSR могут быть использованы ак компонент в построении потоковых шифров с лучшей защитой.

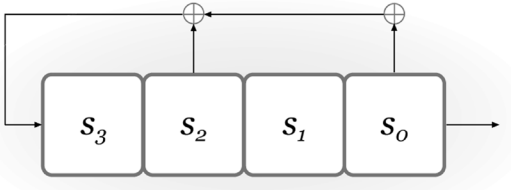

РИСУНОК 6.1 : Регистр сдвига с линейной обратной связью.

LFSR состоит из массива n регистров sn−1, . . . , s_0 вместе с петлей обратной связи, определенной набором из n коэффициентов обратной связи cn−1, . . . , c0. (См. Рисунок 6.1.) Размер массива называется степень LFSR. Каждый регистр хранит одиночный бит, и состояние st LFSR в любой точке времени - это набор битов, хранящихся в регитсрах. Состояние LFSR обновляется каждые несколько «тика часов» путем движения значений во всех регистрах вправо, устанавливая тем самым новое значение самого левого регистра равным XOR подмножества текущих регистров с подмножеством, определенным коэффициентами обратной связи. То есть состояние в какое-то время $s_{n-1}^{(t)}, \ldots, s_0^{(t)}$, затем состояние после следующего тика часов $s_{n-1}^{(t+1)}, \ldots, s_0^{(t+1)}$с

$$s_i^{(t+1)} := s_{i+1}^{(t)}, \qquad i = 0, \ldots, n-2$$
$$s_{n-1}^{(t+1)} := \bigoplus_{i=0}^{n-1} c_i \, s_i^{(t)}.$$

Рисунок 6.1 показывает степень-4 LFSR с $c_0 = c_2 = 1$ и $c_1 = c_3 = 0$.

С каждым тиком часов, LFSR выводит значение самого правого регистра s_0. Если начальное состояние LFSR $s_{n-1}^{(0)}, \ldots, s_0^{(0)}$, первые n бит выходногопотока - ровно $s_0^{(0)}, \ldots, s_{n-1}^{(0)}$. Следующий выходной бит - bit is $s_{n-1}^{(1)} = \bigoplus_{i=0}^{n-1} c_i \, s_i^{(0)}$

Если мы обозначим выходные биты y1, y2, . . ., где yi = s(i−1), тогда

$$y_i = s_{i-1}^{(0)}, \qquad i = 1, \ldots, n$$
$$y_i = \bigoplus_{j=0}^{n-1} c_j \, y_{i-n+j-1} \qquad i > n.$$

В качестве примера, используя LFSR с Рисунка 6.1, если начальное состояние (0, 0, 1, 1), тогда состояния на первые несколько временных периодов

(0, 0, 1, 1)

(1, 0, 0, 1)

(1, 1, 0, 0)

(1, 1, 1, 0)

(1, 1, 1, 1)

и результат (который может быть считан из самой правой колонки выше) - это поток битов 1, 1, 0, 0, 1, ...

Состояние LFSR содержит n бит; таким образом, LFSR может циклически проходить максимум 2n возможных состояния перед повтором. Когда состояния повторяются, выходные биты повторяются, и это означает, что выходная последовательность начнет повторяться после того, как будет сгенерировано максимум 2^n выходных битов. Максимальная длина LFSR проходит циклом через все $2^n - 1$ ненулевые состояния перед повтором. (Обратите внимание, что, если полностью нулевое состояние когда-либо реализуется, LFSR останется в таком состоянии навсегда, поэтому мы и исключили его.) Максимальной ли длины LFSR или нет зависит только от коэффициентов обратной связи; если он максимальной длины, тогда при инициализации в любое ненулевое состояние он будет проходить через все $2^n - 1$ ненулевые состояния. Понятно, как становить коэффициенты обратной связи, чтобы получить максимальную длину LFSR, хотя детали находятся за пределами данной книги.

Атака с восстановлением . Результат LFSR максимальной длины степени n имеет хорошие статистические свойства; например, каждая n-битная строка появляется с относительно равной частотой в выходном потоке LFSR. Тем не менее, LFSR являются не очень хорошими псевдослучайными генераторами для криптографических целей, потому что их результат предсказуем. Это проистекает из того факта, что атакующий может рекоснтруировать целое состояние степени-n LFSR после получения максимум 2n выходных битов. Чтобы в этом убедиться, предположим, что и начальное состояние и коэффициенты обратной связи какого-то LFSR неизвестны. Первые n выходных бит y1, . . . , yn LFSR точно отражают начальное сотояние. Учитывая следующие n выходных бит yn+1, . . . , y^{2n}, атакующий может установить систему n линейных уравнений в n неизвестных c0, . . . , cn−1:

$y_{n+1} = c_{n-1}\, y_n \oplus \cdots \oplus c_0\, y_1$

$y_{2n} = c_{n-1}\, y_{2n-1} \oplus \cdots \oplus c_0\, y_n.$

Можно показать, что вышеописанные уравнения линейно независимые (по модулю 2) для LFSR максимальной длины, и таким образом, однозначно установить коэффициенты обратной связи. (Решение может быть с легкостью найдено при помощи линейной алгебры.) При известных коэффициентах обратной связи все последующие выходные биты LFSR могут быть легко вычислены.

6.1.2 Добавление нелинейности

Линейные отношения между выходными битами LFSR - это то, что упрощает атаку. Чтобы предотвратить такую атаку мы должны ввести некоторую нелинейность, то есть некоторые операции отличные от XOR. Существует несколько различных подходов для этого, но мы здесь изучим только некоторые из них.

Нелинейная обратная связь. Очевидный способ изменить LFSR является - это сделать петлю обратной связи нелинейной. Регистр сдвига с нелинейной обратной связью (FSR) будет состоять из массива регистров, каждый из которых содержит одиночный бит. Как и ранее, состояние FSR обновляется каждые несколько «тика часов» путем движения значений во всех регистрах вправо; теперь, однако, новое значение самого левого регистра является нелинейной функцией текущих регистров. Другими словами, если состояние в какой-то промежуток времени t - $s_0^{(t)}, \ldots, s_{n-1}^{(t)}$, тогда состояние после следующего тика часов - $s_0^{(t+1)}, \ldots, s_{n-1}^{(t+1)}$ с

$$s_i^{(t+1)} := s_{i+1}^{(t)}, \qquad i = 0, \ldots, n-2$$

$$s_{n-1}^{(t+1)} := g(s_0^{(t)}, \ldots, s_{n-1}^{(t)})$$

для какой-то нелинейной функции g. Как и ранее, FSR выводит значение самого правого регистра s0 при каждом тике часов.

Есть возможность разработать нелинейные FSR с максимальной длиной и так, чтобы результат имел хорошие статистические свойства.

Генераторы нелинейных комбинаций. Еще один подход - это ввести нелинейность в выходную последовательность. В самом простом случае мы можем иметь LFSR, как и ранее (где новое значение самого левого регистра вычисляется как линейная функция текущих регистров), но где результат при каждом тике часов является нелинейной функцией g всех текущих регистров, а не просто самым правым регистром. Здесь важно то, что g будет сбалансирована в том смысле, что $Pr[g(s_0, \ldots, s_{n-1}) = 1] \approx 1/2$ (где вероятность - это более единообразный выбор $s_0, \ldots, s_{n-1}$); иначе, хотя должно быть тяжело реконструировать целое состояние LFSR на основе результата, выходной поток отклонится и, таким образом, будет более отличим от универсального.

Вышеописанный вариант заключается в том, чтобы использовать несколько LFSR (с индивидуальным выходным потоком для каждого, вычисленным, как и ранее, просто взяв значении самого правого регистра каждого LFSR) и сгенерировать реальный выходной поток путем комбинирования результата индивидуальных LFSR каким-то нелинейным способом. Это даст то, что известно как (нелинейный) генератор комбинаций. Индивидуальные LFSR не обязательно должны иметь одинаковую степень, и, фактически, длина цикла генератора комбинаций будет максимизирована, если у них не одинаковая степень. Здесь внимательно нужно следить, чтобы убедиться, что выходной поток генератора комбинаций не очень-то коррелируется с любым другим из выходных потоков индивидуальных LFSR; высокая корреляция может привести к атакам на индивидуальные LFSR, таким образом подрывая возможность использования нескольких LFSR в конструкции.

6.1.3 Тривиум

Чтобы проиллюстрировать идеи из предыдущего раздела мы вкратце опишем потоковый шифр Тривиум. Потоковый шифр был выбран в качестве части портфеля проекта eSTREAM, европейской инициативы 2008 года, цель которой была идентификация новых потоковых шифров. Тривиум был разработан с небольшим описанием и компактной аппаратной реализацией.

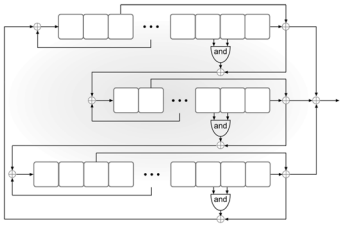

РИСУНОК 6.2 : Сх ематическая иллюзстранция шифра Тривиум с (сверху вниз) тремя спаренными нелинейными FSR A, B и C.

Тривиум использует спаренные нелинейные FSR, обозначенные как A, B и C , а также имеет степень 93, 84 и 111, соответственно. (См. Рисунок 6.2.) Состояние st шифра Тривиум - это просто 288 бит, содержащие значения во всех регистрах этих FSR. Алгоритм GetBits для шифра Тривиум работает следующим образом: На каждом тике часов вывод каждого из FSR - это XOR его самого правого и одного дополнительного регистра; вывод шифра Тривиум - это XOR вывода битов трех FSR. FSR являются спаренными : при каждом тике часов новое значение самого левого регистра каждого FSR вычисляется в качестве функции одного из регистров в том же FSR и подмножества регистров второго FSR. (Функция обратной связи для A зависит от одного регистра A и четырех регистров C; функция обратной связи для B зависит от регистра B и четырез регистров A; и функция обратной связи для C зависит от одного регистра C и четырех регистров B.) Функция обратной связи в каждом лсучае является нелинейной.

Алгоритм Init шифра Тривиум принимает 80-битный ключ и 80-битный IV . Ключ загружается в 80 самых левых регистров A, а IV загружается в 80 самых левых регистров B. Оставшиеся регистры устанавливаются на 0, кроме трех самых правых регистров C, которые устанавливаются на 1. Затем GetBits запускается $4 \cdot 288$ раз (с отброшенным выводом), и итоговое состояние берется как st_0.

На сегодняшний день нет известных криптографических атак лучше, чем полный перебор ключей, действующих против полного шифра Тривиум.

6.1.4 RC4

LFSR эффективны, когда внедрены в оборудование, но имеют низкую производительность в программном обеспечении. По этой причине были изучены альтернативные разработки потоковых шифров. Прекрасный пример - это RC4, который был разработан Роном Ривестом (Ron Rivest) в 1987 году. RC4 примечателен своей скоростью и простотой, а также выдерживал серьезную атаку много лет. Он сегодня широко применяется, и поэтому мы опишем его; предупреждаем читателя, однако, что самые новые атаки показали серьезные криптографические слабости RC4 и больше его использовать нельзя.

Состояние RC4 - это 256-байтный массив S, который всегда содержит перестановку элементов 0, . . . , 255 вместе с двумя значениями $i, j \in \{0, \ldots, 255\}$. Алгоритм Init для RC4 представлен как Алгоритм 6.1. Для упрощения мы возьмем что 16-байтный (128-битный) ключ k, хотя алгоритм может обрабатывать ключи длиной от 1 байта до 256. Мы обозначим байты S как S[0], . . . , S[255], а байты ключа как k[0], . . . , k[15].

АЛГОРИТМ 6.1

Алгоритм Init для RC4

Вход: 16-байтный ключ k
Выход: Начальное состояние (S, i, j)
(Обратите внимание: Все сложение делается по модулю 256)
for i = 0 to 255:
S[i] := i
k[i] := k[i mod 16]
j := 0
for i = 0 to 255:
j := j + S[i] + k[i] Swap S[i] and S[j]
i := 0, j := 0
return (S, i, j)

Во время инициализации S является первым набором для тождественной перестановки (то есть с S[i] = i для всех i) и k расширяется до 256 байт путем повторения. Затем каждый элемент S меняется местами как минимум один раз с другим элементом S в «псевдослучайном» расположении. Индексы i, j устанавливаются на 0, и (S, i, j) выводится как начальное состояние .

Состояние затем используется для генерирования последовательности выходных битов, как показано в Алгоритме 6.2. Индекс i просто инкрементируется (по модулю 256), а j изменяется каким-либо «псевдослучайным» способом. Элементы S[i] и S[j] меняются местами, и выводится значение S на позиции S[i] + S[j] (вновь вычисленное по модулю 256). Обратите внимание, что каждый элемент S меняется местами с каким-либо другим элементом S (возможно с самим собой) как минимум один раз каждые 256 итераций, обеспечивая тем самым хорошеее «перемешивание « перестановки S.

АЛГОРИТМ 6.2

Алгоритм GetBits для RC4

Вход: Текущее состояние (S, i, j)

Выход: Выходной байт y; обновленное состояние (S, i, j) (Обратите внимание: Все сложение делается по модулю 256)

i := i + 1
j := j + S[i]
Swap S[i] and S[j]
t := S[i] + S[j]
y := S[t]
return (S, i, j), y

RC4 не был предназначен для приема IV в качестве входных данных; однако, на практике IV часто принимается посредством простого соединения его с

действующим ключом kr перед инициализацией. То есть выбирается случайный IV желаемой длины, k устанавливается так, чтобы равняться соединенным IV и kr (это можно осуществить путем присоединения IV как в начало, так и в конец), затем Init запускается, как в Алгоритме 6.1 для генерирования начального состояния. Затем производятся выходные биты с использованием Алгоритма 6.2, точь в точь как ранее. Предполагая, что RC4 используется в несинхронизированном режиме (см. Раздел 3.6.1), IV тогда будет отправлен в открытую получателю, который предположительно уже имеет действующий ключ kr, таким образом, позволяя им сгенерировать такое же начальное состояние и, соответственно, такой же выходной поток. Данный метод включения IV используется в Wired Equivalent Privacy (WEP), стандарте шифрования для защиты связи в беспроводных сетях 802.11.

Следует задуматься о таком относительно узконаправленном способе именения RC4 для принятия IV. Даже если RC4 был бы защищенным потоковым шифром при использовании (только) ключа, как изначально и планировалось, нет причин полагать, что он должен быть защищенным после изменения для использования IV таким образом. В действительности,в отличие от ключа, IV открыт для атакующего (так как отсылается в открытую; более того, использование различных IV с одним и тем же фиксированным ключом kr, как было бы при использовании RC4 в несинхронизированном режиме, означает, что соотвествующие значения k используются для инициализации состояния RC4. Как мы увидим далее, обе эти проблемы ведут к атакам, когда RC4 используется таким образом.

Атаки на RC4. Хотя RC4 широко распространен в современных системах, различные атаки на RC4 были известны на протяжении нескольких лет. Из-за этого RC4 больше не следует использовать; вместо него следует использовать более новый потоковый шифр или блочный шифр.

Мы начнем с демонстрации простой статистической атаки на RC4, которая не опирается на доверенные стороны, которые используют IV . Атака эксплуатирует тот факт, что второй выходной байт RC4 (слегка) отклонен к 0. Пусть St обозначает состояние массива S после t итераций GetBits с S0 , обозначающим начальное состояние. Рассматривая S_0 (эвристически) как универсальную перестановку $\{0, \ldots, 255\}$ с вероятностью $1/256 \cdot (1 - 1/255) \approx 1/256$, справедливо, что $S_0[2] = 0$ и $X \stackrel{\text{def}}{=} S[1]$ $f = 2$. Предположим на мгновение, что это наш случай. При первой итерации GetBits значение i инкрементируется до 1, и j устанавливается на $S_0[i] = S_0[1] = X$. Затем $S_0[1]$ и $S_0[X]$ меняются местами так, что в конце итерации мы имеем $S_1[X] = S_0[1] = X$. При второй итерации i инкрементируется к 2, и j приписывается значение

$$j + S_1[i] = X + S_1[2] = X + S_0[2] = X,$$

так как $S_0[2] = 0$. Затем $S_1[2]$ и $S_1[X]$ меняются местами так, что $S_2[X] = S_1[2] = S_0[2] = 0$ и $S_2[2] = S_1[X] = X$. Наконец, выводится значение S_2 на позиции

$S_2[i]+S_2[j] = S_2[2]+S_2[X] = X$; это и есть значение $S_2[X] = 0$. Когда $S_0[2]$ $f= 0$, второй выходной байт единообразно распределен. Таким образом, вероятность того, что второй выходной байт - это 0 равняется

$$\Pr[S_0[2] = 0 \text{ and } S_0[1] \neq 2] + \frac{1}{256} \cdot \Big(1 - \Pr[S_0[2] = 0 \text{ and } S_0[1] \neq 2]\Big)$$
$$= \frac{1}{256} + \frac{1}{256} \cdot \left(1 - \frac{1}{256}\right) \approx \frac{2}{256},$$

или вдвое больше, что было бы прогнозируемо для универсального значения.

Сам по себе вышеописанный пример может и не рассматриваться как серьезная атака, хотя, кажется, он действительно указывает на внутреннюю проблему RC4. Более серьезная атака на RC4 вполне возможна, если IV водится путем добавления его в начало ключа. Такая атака может быт использована, чтобы извлечь ключ, независимо от его длины, и это реально намного серьезнее, чем дифференциальная атака, наподобие той, что была описана выше. Важно отметить, что такая атака может быть использована, чтобы полностью взломать стандарт шифрования WEP, упомянутый ранее, и она сделала много, чтобы стандарт был заменен.

Ядро атаки - это способ расширить знание о первых n байтах k до знания о первых (n + 1) боайтах k. Обратите внимание, что если IV прикреплен в начало действующего ключа kr (поэтому $k = IV \| k^r$), первые несколько байт k отдаются атакующему даром! Если IV длиной n байт, тогда злоумышленник может использовать эту атаку, чтобы сначала извлечь (n + 1) байт ключа k (который является первым байтом настоящего ключа kr), затем следующий байт ключа k и так далее, пока не извлечет весь ключ.

Предположим, что IV длиной 3 байта, ка в случае с WEP. Атакующий ждет, пока первые два байта IV не примут специфическую форму. Атака может быть произведена с несколькими вариантами для первых двух байтов IV , но мы рассматриваем случай, когда IV принимает форму IV = (3, 255, X) для X произвольного байта. Это означает, коненчо же, что k[0] = 3, k[1] = 255, и k[2] = X в Алгоритме 6.1. Можно проверить, что после первых четырех итераций второго витка Init, у нас будет

$$S[0] = 3, S[1] = 0, S[3] = X + 6 + k[3]. \qquad (6.1)$$

При следующих 252 итерациях алгоритма Init, i всегда будет больше 3. Поэтому значения S[0], S[1] и S[3], соответственно, не будут изменены, так как j никогда не примет значения 0, 1 или 3. Если мы (эвристически) рассмотрим j как принимающее универсальное значение при каждой итерации, это будет означать, что S[0], S[1] и S[3], соответствнно, не будут меняться с вероятностью $(253/256)^{252} \approx 0.05$ или 5% времени. Предполгая, что дело обстоит именно так, первый байт, выведеный GetBits будет S[3] = X + 6 + k[3]; поскольку X известен,

это открывает k[3].

Итак, атакующий знает, что 5% времени первый байт вывода коррелируется с k[3], как и описано выше. (Это уже намного лучше, чем случайное гадание, которое верное 1/256 = 0.4% времени.) Таким образом, собирая достаточно много образцов первого байта вывода, для нескольких IV , имеющих правильную форму, атакующий получает высокую достоверность возможного значения для k[3].

Блочные шифры.

Вспомним из Раздела 3.5.1, что блочный шифр является эффективной ключевой перестановкой $F : \{0, 1\}^n \times \{0, 1\}^A \rightarrow \{0, 1\}^A$. Это означает, что функция Fk, определенная $F(x) \stackrel{def}{=} F(k, x)$, является биекцией (то есть перестановкой), и, более того, F и ее обратная величина F^{-1} являются эффективно вычисляемыми, если имеется k. Мы будем называть n - длиной ключа и A - длиной блока F, и здесь мы явно позволим им отличаться. Длина ключа и длина блока теперь фиксированные константы, тогда как в Главе 3 они рассматривались как функции параметра защиты. Это ставит нас в условие конкретной защиты, а не асимптотической. 2 Требования конкретной защиты для блочных шифров довольно жесткие, и блочный шифр, в общем, считается всего лишь «хорошим», если наиболее известная атака (с предварительной обработкой) имеет временные трудности, сравнимые с атакой с использованием «грубой силы» для поиска ключа. Таким образом, если шифр с длиной ключа n = 256 может быть взломан за время 2^{128}, шифр (в общем) считается незащищенным, даже если атака продолжительностью 2^{128} все еще невозможна. В то же время, в асимптотических условиях атака сложностью $2^{n/2}$ не считается эффективной, пока для нее требуется экспоненциальное временя (следовательно, шифр, для которого такая атака возможна, все еще может удовлетворять определение псевдослучайно перестановки). В конкретных условиях, однако, мы должны предупредить о реальной сложности атаки (а не о ее асиптотическом поведении). Более того, есть проблема, что существование такой атаки может указывать на более фундаментальную слабость в конструкции шифра.

Блочные шифры разработаны, чтобы вести себя, по крайней мере как (сильные) псевдослучайные перестановки; см. Определение 3.28. Моделирование блочных шифров в виде псевдослучайных перестановок позволяет добиться доказательств защиты для конструкций, на основе блочных шифров, а также открывают необходимые требования блочных шифров. Четкое понимание того, какую цель блочные шифры должны выполнять, играет важную роль в их раз

2Хотя блочный шифр с ключом фиксированной длины не имеет «параметра защиты», мы все еще рассматриваем защиту как такую, что зависит от длины ключа, и, таким образом, выражаем это значение как n.

работке. Точка зрения о том, что блочные шифры должны моделироваться как псевдослучайные перестановки, служила, по крайней мере в недалеком прошлом, основой их разработки. В качестве примера, на конкурсе проектов касательно нового стандарта Advanced Encryption Standard (AES), с которым мы столкнемся позже в этой главе, прозвучали следующие критерии оценки:

Защита, обеспеченная алгоритмом, является наиболее важным фактором.... Алгоритмы будут оцениваться по следующим факторам...

• Степень, в которой алгоритм неотличим от случайной перестановки...

Современные блочные шифры подходят для всех конструкций, включая псевдослучайные перестановки (или псевдослучайные шифры), которые мы видели в данной книге.

Часто блочные шифры разрабатываются (и предполагаются) для удовлетворения даже более сильных защитных свойств, как мы и писали в Разделе 6.3.1.

Несмотря на тот факт, что блочные шифры сами по себе не являются схемами шифрования, стандартная терминология для атак на блочный шифр F такова:

• При атаках на основе открытого текста, атакующий получает пары входных /выходных данных

{(xi, Fk (xi))} (для неизвестного ключа k), с {xi} за пределами контроля атакующего.

• При атаках на основе подобранного открытого текста, атакующий получает {Fk (xi)} (снова же, для неизвестного ключа k) для серии входных данных {xi}, подобранных атакующим.

• При атаках на основе подобранного шифротекста, атакующий получает {Fk(xi)} для {xi}, подобранного атакующим, равно как и $\{F^{-1}(yi)\}$ для подобранного {yi}.

Рассмотрение длины ключа в качестве параметра имеет смысл, при сравнении блочных шифров с разной длиной ключей или при использовании блочного шифра, который поддерживает ключи разной длины.

Кроме использование вышеописанного для отличия Fk от универсальной перестановки, мы также будем заинтересованы в атаках с восстановлением ключа , при которых атакующий имеет возможность извлечь ключ k после взаимодействия с F_k . (Это сложнее, чем иметь возможность отличить Fk от универсальной.)

Согласно таксономии, псевдослучайная перестановка не может быть отличима от универсальной перестановки во время атаки на основе подобранного открытого текста, так как сильная псевдослучайная перестановка не может выделяться даже при атаке на основе подобранного шифротекста.

Подстановочно-перестановочные сети

Блочный шифр должен вести себя как случайная перестановка. Существует 2^A !

перестановок A-битных строк, поэтому представление произвольной перестановки с A-битной длиной блока требует $\log(2^A!) \approx A \cdot 2A$ бит. Это непрактично для $A > 20$ и невозможно для $A > 50$. (Забегая вперед, современные блочные шифры имеют длину блока $A \geq 128$.) Трудность при разработке блочного шифра состоит в том, чтобы построить набор перестановок с компактным описанием (а именно, коротким ключом), который ведет себя как случайная перестановка. В частности, все прямо как при вычислении случайной перестановки при двух входных элементах разница, лишь в том, что одиночный бит должен дать два (почти) независимых выходных элемента (они не могут быть полностью независимы, так как они не могут быть равны), при этом также изменение одного из битов выходного элемента на $Fk(\bullet)$, где k является универсальным и неизвестным атакующему, должно дать (почти) независимый результат. Подразумевается, что одно-битовое изменение во входном элементе должно «повлиять» на каждый бит выходного элемента. (Обратите внимание, что это не значит, что все выходные биты будут изменены, это было бы другим поведением, которое можно было бы спрогнозировать для случайно перестановки. Скорее, мы лишь неформально имеем в виду, что каждый бит выходного элемента изменится, с приблизительно половинной вероятностью.) Для этого необходимо будет проделать кое-какую работу. 2^{128}

Парадигма смешения-рассеивания. В дополнение к своей работе об идеальной стойкости, Шеннон (Shannon) также представил базовую парадигму конструирования компактных, псевдослучайных перестановок. Основной идеей является конструирование псевдослучайных перестановок F с большой длиной блоков из множества более мелких случайных (или псевдослучайных) перестановок $\{f_i\}$, с маленькой длиной блока. Давайте посмотрим, как это работает на базовом уровне. Скажем, мы хотим, чтобы длина блока F была 128 бит. Мы можем определить F следующим образом: ключ k для F определит 16 перестановок $f_1, \ldots, f_{16}$, каждая из которых имеет длину блока в 8 бит (1 байт). [3] Учитывая входные данные $x \in \{0, 1\}$, мы разбираем ее на 16 байтов $x_1 \bullet \bullet \bullet x_{16}$ и затем устанавливаем

$$F_k(x) = f_1(x_1) \| \bullet \bullet \bullet \| f_{16}(x_{16}). \quad (6.2)$$

Эти раундовые функции $\{f_i\}$ должны ввести смешение в F.

Необходимо немедленно прояснить, однако, что F, как определено выше, не будет псевдослучайной. В частности, если x и x^r отличаются только первым битом, тогда $F_k(x)$ и $F_k(x^r)$ будут отличаться только первым байтом (независимо от ключа k). В то же время, если F была бы истинно случайной перестановкой,

---

[3] Произвольная перестановка на 8 битах может быть представлена с использованием $\log(28!) \approx 1600$ бит, поэтому длина ключа для F составит $16 \cdot 1600$ бит или 3 килобайта. Это намного меньше, чем $\approx 128 \cdot 2128$ бит, которые понадобились бы для определения произвольной перестановки на 128 битах.

тогда изменение первого бита входных данных, возможно, повлияло бы на все байты результата.

По этой причине вводится шаг рассеивание , когда биты результата переставляются или «смешиваются», используя смешивающую перестановку. Это дает эффект распространения локального изменения (например, изменение первого байта) по всему блоку. Шаги смешения/рассеивания, вместе называемые раундовые, повторяются множество раз. Это помогает обеспечить изменение одиночного бита входных данных, что повлияет на все биты результата.

В качестве примера двухраундовый блочный шифр согласно данному подходу будет работать следующим образом. Сначала вводится смешение путем вычисления промежуточного результата $f_1(x_1)\| \bullet \bullet \bullet \| f_{16}(x_{16})$, как в Уравнении (6.2). Биты результата, таким образом, «перетасовываются» или переорганизовываются, чтобы дать xr. Затем вычисляется $f_1{}^r(x^r) \| \bullet \bullet \bullet \| f^r(x^r)$ (где $xr = xr \bullet \bullet \bullet xr$), используя, возможно, различные функции f r, и биты результата переставляются, чтобы выдать xrт. $\{f_i\}$, $\{f^r\}$ и смешивающая(-ие) перестановка(-и) могли бы быть случайными и зависимыми от ключей, как мы описали выше. На практике, однако, они специально разрабатываются и фиксируются, а ключ вводится различными способами, как мы опишем ниже.

Подстановочно-перестановочные сети. Сеть смешения-рассеивания (SPN) может рассматриваться как прямая имплементация парадигмы смешения/рассеивания. Разница в том, что сейчас раундовые функции имеют определенную форму , а не просто выбираются из множества всех возможных перестановок в какой-то области. В частности, вместо того, чтобы иметь ключ (часть ключа) k, определять перестановку f, мы зафиксируем публичную «заместительную функцию» (то есть перестановку S под названием S-блок, а затем позволим k определить функцию f , данную $f(x) = S(k \oplus x)$.

Чтобы узнать, как это работает конкретно, рассмотрим SPN с 64-битной длиной блока на основе коллекции 8-битных (1-байтных) S-блоков $S_1, \ldots, S_8$. (См. Рисунок 6.3.) Расчет шифра происходит в несколько раундов, когда в каждом раунде мы применяем следующую последовательность операций к 64-битному входному элементу x текущего раунда (входной элемент первого раунда также является входным элементом шифра):

1. Смешение ключей: Зададим $x := x \oplus k$, где k является подключом текущего раунда;

2. Замена: Зададим $x := S_1(x_1)\| \bullet \bullet \bullet \| S_8(x_8)$, где xi - это байт i от x;

3. Перестановка: Переставим биты x , чтобы получить выходной результат раунда.

Выходной результат каждого раунда подается как входные данные следующего раунда. После последнего раунда следует последний шаг смешения клю-

чей, результатом которого является выхоной элемент шифра. (По принципу Керчхоффа (Kerckhoffs) мы предположим, что S-блоки и смешивающая(-ие) перестановка(-и) являются публичными и известными любому атакующему. Это значит, что без последнего шага смешения ключей , последние шаги по замене и перестановке не дадут никакой дополнительной защиты, поскольку они не зависят от ключа.) Рисунок 6.4 показывает

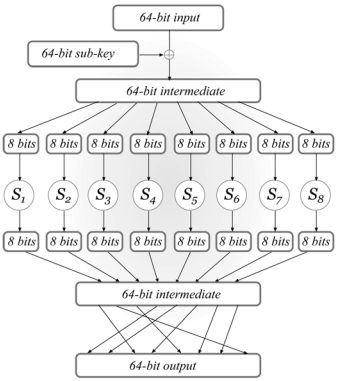

РИСУНОК 6.3: Один раунд подстановочно-перестановочной сети.

высокоуровневую структуру SPN с 16-битной длиной блока и другой набор 4-битных S-блоков, используемых в каждом раунде.

Различные подключи (или раундовые ключи) используются в каждом раунде. Действующий ключ блочного шифра иногда называется мастер-ключ. Раундные подключи происходят от мастер-ключа гласно расписанию смены ключей. Расписание смены ключей часто простое и может работать путем простого приема подмножеств битов мастер-ключа, хотя более сложные расписания также могут быть определены. r-раундовая SPN имеет r (полных) раундов смешения ключей, замен S-блоков и применений смешивающей перестановки, за которой следует последний шаг смешения ключей. (Это означает, что в r-раундовой SPN используется r + 1 подключей.)

Любая SPN обратима (получает ключ). Чтобы убедиться в этом мы покажем, что получив результат SPN и ключ, есть возможность извлечь входные данные. Достаточно будет показать, что один раунд может быть обратимым; из этого следует, что вся SPN может быть обратима и работать от последнего раунда назад к началу. Но обратить один раунд легко: смешивающая перестановка может с легкостью быть обратной, потому что это просто реорганизация битов. Поскольку S-блоки являются перестановками (то есть взаимооднозначными), они также могут быть обратными. Результаты затем могут быть сложены по модулю 2 с соответствующим подключом, чтобы получить оригинальный входной элемент. Таким образом:

ПРЕДПОЛОЖЕНИЕ 6.3 *Пусть F будет ключевой функцией, определенной SPN, в которой все S-блоки являются перестановками. Тогда независимо от расписания смены ключей и количества раундов, Fk является перестановкой для любого k.*

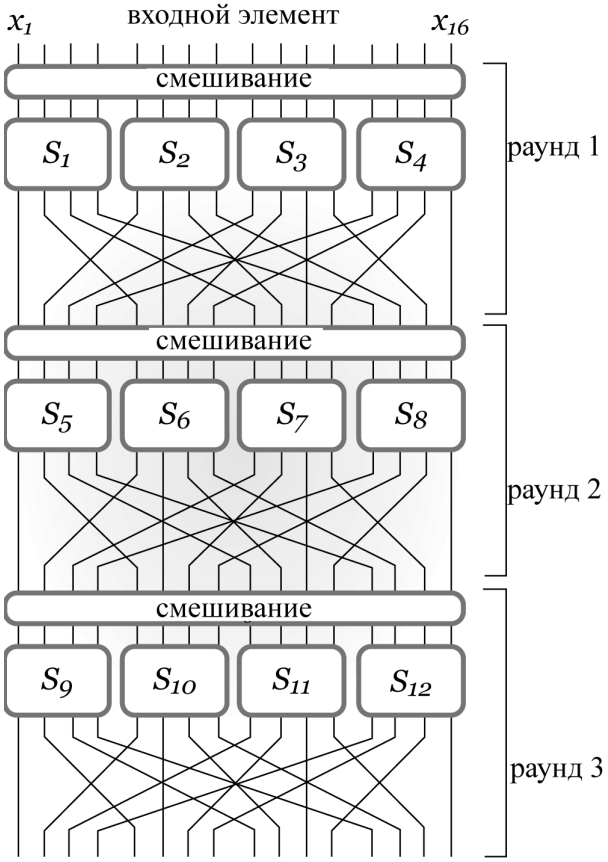

РИСУНОК 6.4: Подстановочно-перестановочная сеть.

Количество раундов вместе с точным выбором S-блоков, смешивающих перестановок и расписания смены ключей - это то, что обязательно определяется независимо от того, является ли данный блочный шифр тривиально хрупким или надежно защищенным. Теперь обсудим основной принцип разработки S-блоков и смешивающих перестановок.

Лавинный эффект. Как неоднократно говорилось, важным свойством лю-

бого блочного шифра является то, что изменение входного элемента должно «повлиять» на каждый бит результата. Мы назовем это явление лавинным эффектом. Один из способов вызвать лавинный эффект в подстановочно-перестановочной сети - это обеспечить наличие следующих двух свойств (при максимальном количестве раундов):

1. S-блоки разработаны так, чтобы, изменяя один бит входного элемента, S-блок меняет как минимум два бита в выходном элементе S-блока.

2. Смешивающие перестановки разработаны так, чтобы выходные биты любого доступного S-блока использовались как входные элементы в множественные S-блоки в следующем раунде.

Чтобы увидеть, как это производит лавинный эффект, по крайней мере эвристически, предположим, что S-блоки все такие, что изменение одного бита входного элемента S-блока выливается в изменение ровно двух битов выходного элемента S-блока, и что смешивающие перестановки выбраны, как указано выше. Для конкретики, предположим, что S-блоки имеют входной/выходной размер в 8 бит, и что длина блока шифра - 128 бит. Рассмотрим сейчас, что случится, когда блочный шифр примениться к двум входным элементам, которые отличаются одним битом:

1. После первого раунда промежуточные значения отличаются ровно в двух положениях битов. Это потому что сложение по модулю 2, текущего ключа сохраняет 1-битную разницу в промежуточных значениях, и поэтому входные элементы во все S-блоки, кроме одного, - идентичны. В одном S-блоке, в котором входные элементы отличаются, выходной результат S-блока показывает 2-битную разницу. Смешивающая перестановка, применяемая к результатам, меняет позиции эти изменений, но сохраняет 2-битную разницу.

2. Смешивающая перестановка, применяемая в конце первого раунда, распространяет две позиции битов, в которых промежуточные результаты отличаются, в два различных S-блока во втором раунде. Это остается истинным, даже после того, как соответствующий ключ складывается по модулю 2 с результатом предыдущего раунда. Итак, во втором раунде у нас теперь два S-блока, которые получили входные элементы, отличающиеся на один бит. Таким образом, в конце второго раунда промежуточные значения отличаются на 4 бита.

3. Продолжая тот же аргумент, мы прогнозируем 8 бит промежуточного значения после третьего раунда, 16 бит - после четвертого раунда и все 128 бит выходного результата в конце седьмого раунда.

Последняя точка не очень точная, и определенно есть вероятность, что будет некоторые отличия результатов какого-либо из раундов от прогнозируемых. (Фактически, так и должно быть, потому что результаты не должны отличаться по всем битам.) Однако, вышеописанный анализ, дает нижнюю границу по количеству

раундов: если используется менее 7 раундов, тогда должен быть некий набор выходных битов, которые не затронет однобитное изменение во входных данных, вызвав тем самым возможность распознать шифр из случайной перестановки.

Можно ожидать, что «наилучшим» способом разработать S-блоки было бы выбрать их случайным образом (учитывая то ограничение, что они являются перестановками). Интересно, что это нам не подходит, по крайней мере, если мы хотим удовлетворить коректный критерий, упомянутый ранее. Рассмотрим случай, когда S-блок, работающий с 4-битными входными данными, и пусть x и xr будут двумя разными значениями. Пусть y = S(x), и теперь рассмотрим выбор универсального yr f= y в качестве значения S(xr). Имеется 4 строки, отличающиеся от y только лишь одним битом, и поэтому с вероятностью 4/15 мы выберем yr , которое не отличается от y на два или более бита. Проблемаусугубляется, если принять во внимание все пары входных элементов, которые отличаются на один бит.

Сделаем вывод, основанный на этом примере, что, как правило, S-boxes должны быть тщательно разработы, а не просто выбраны случайным образом. Случайные S-блоки - это также нехорошо для защиты от атак, как и блоки, которые мы покажем в Разделе 6.2.6.

Если блочный шифр должен также быть строго псевдослучайным, тогда лавинный эффект должен обязательно применяться к его обратному варианту. То есть изменение одного бита выходного элемента должно влиять на каждый бит входного элемента. Поэтому это очень полезно, если S-блоки разрабатываются так, чтобы, меняя один бит выходного элемента S-блока, менялись по крайней мере два бита входного элемента в S-блоке . Достижение лавинного эффекта в обоих направлениях - это еще одна причина для дальнейшего увеличения количества раундов.

Атаки на SPN с уменьшенным количеством раундов

Опыт и многие годы криптографической деятельности показывают, что подстановочно-перестановочные сети являются хорошим выбором для построения псевдослучайных перестановок, если только особое внимание уделялось выбору S-блокам, смешивающим перестановкам и расписанию смены ключей. Advanced Encryption Standard, описанный в Разделе 6.2.5, аналогичный по структуре подстановочно-перестановочной сети, описанной выше, и бытует мнение, что является сильной псевдослучайной перестановкой.

Сила шифра F, построенного таким образом, зависит в значительной степени от количества раундов. Для того, чтобы узнать больше о сути подстановочно-перестановочных сетей, мы продемонстрируем атаки на SPN, в которых очень мало раундов. Эти атаки прямолинейные, но обязательны к ознакомлению, так как демонстрируют, почему необходимо большое число раундов.

Тривиальный случай. Для начала рассмотрим тривиальный случай, в котором

F состоит из одного полного раунда и последнего шага смешения ключей. Мы покажем, что злоумышленник, имея только одну входную/выходную пару (x, y), может с легкостью узнать секретный ключ k, для которого $y = F_k(x)$. Злоумышленник начинает с выходного значения y и затем инвертирует смешивающую перестановку и S-блоки. Он это может сделать, как было отмечено ранее, потому что полная спецификация смешивающей перестановки и S-блоков находится в публичном доступе. Промежуточное значение, которое вычисляет злоумышленник, - именно $x \oplus k$ (предполагая, без потери обобщенности, что мастер-ключ используется как подключ в единственном раунде сети). Так как злоумышленник также знает входной элемент x, он может тут же извлечь секретный ключ k. Таким образов, мы наблюдаем полный взлом.

Хотя это и тривиальная атака, она демонстрирует , что в любой подстановочно-перестановочной сети отсутствует защита, полученная от выполнения перестановок S-блоками или применения смешивающей перестановки после последнего шага смешения паролей.

Атака на одноraундовую SPN. Теперь у нас есть полный раунд, за которым следует шаг смешения ключей. Для конкретики, мы предположим 64-битную длину блока и S-блоки с входной/выходной длиной в 8 бит (1 байт). Мы предположим, что 64-битные подключи k_1, k_2 используются для двх шагов смешения ключей, поэтому мастер-ключ $k_1 \| k_2$ SPN будет длиной 128 бит.

Первое наблюдение заключается в том, что мы можем расширить атаку из тривиального случая, описанного выше, чтобы получилась атака с восстановлением ключа, выполнив работы намного меньше, чем 2^{128} . Идея такова: Получив одну входную/выходную пару (x, y), как и ранее, атакующий перебирает все возможные значения для подключа второго раунда k^2.

Для каждого такого значения атакующий может инвертировать последний шаг смешения ключей, чтобы получить вероятное промежуточное значение y^r. Мы увидели выше, что, имея входной элемент x и выходной элемент y^r (целого) раунда SPN, возможный уникальный подключ k1 может быть с легкостью идентифицирован. Таким образом, для каждого возможного выбора k_2 атакующий извлекает соответствующий уникальный k1 , для которого $k_1 \| k_2$ может быть мастер-ключом. Следовательно, атакующий получает (за время 2^{64}) список из 2^{64} вариантов мастер-ключа.

Его можно сузить с использованием дополнительной входной/выходной пары за дополнительное время 2^{64} приблизительно; см. также ниже.

Возможно провести еще лучшую атаку, заметив, что индивидуальные биты выходного элемента зависят только от мастер-ключа. Зафиксируем полученную входную/выходную пару (x, y), как и ранее. Теперь злоумышленник будет перебирать все возможные значения первого байта , принадлежащего k_2. Он может сложить по модулю 2 каждое такое значение с первым байтом y , чтобы полу-

чить вероятное значение выходного элемента первого S-блока. Инвертируя этот S-блок, атакующий узнает вероятное значение входного элемента в этот S-блок. Так как входной элемент в этот S-блок является XOR из 8 бит x и 8 бит k_1 (где позиции этих битов зависят от смешивающей перстановки первого раунда и известны атакующему), это дает вероятное значение для 8 бит k_1.

Подводя итого, отметим, что для каждого вероятного значения первого байта k_2 существует вероятное соотвествующее уникальное значение для каких-либо 8 бит k_1. Иными словами, это означает, что для каких-то 16 бит мастер-ключа атакующий уменьшил количество вероятных значений для битов от 2^{16} до 2^8. Атакующий может свести все те вероятные значения за время 2^8. Это можно повторить для каждого байта из k_2, давая 8 списков, каждый из которых содержит 2^8 значений, которые вместе характеризуют возможные значения всего мастер-ключа. Атакующий, таким образом, уменьшил количество возможных мастер-ключей до $(2^8)^8 = 2^{64}$, как и в предыдущей атаке. Общее время, затраченное на все это, теперь $8 \cdot 2^8 = 2^{11}$, что есть резкое улучшение.

Атакующий может использовать дополнительную входную/выходную пару, чтобы еще больше уменьшить пространство возможных ключей. Рассмотрим лист с 28 вероятными значениями какого-то набора из 16 бит мастер-ключа. Атакующему известно, что правильное значение из этого списка должно согласовываться с любой из дополнительных входных/выходных пар, которые известны атакующему. Эвристически, любое неправильное значение из списка согласуется с какой-то дополнительной входной/выходной парой (x^r , y^r) вероятностью, которая не лучше, чем случайное гадание; так как 16-битное значение из таблицы может быть использовано для вычисления 1 байта выходного элемента при наличии входного элемента xr, мы прогнозируем, что неправильное значение будет согласовываться с реальным действующим выходным элементом yr с вероятностью 2−8. Маленькое количество дополнительных входных/выходных пар, таким образом, помогает сузить все таблицы до одного значения в каждой, и на этом этапе мастер-ключ уже можно считать найденным.

Отсюда необходимо извлечь важный урок. Атака возможна, если разные части ключа могут быть изолированы от других частей. Таким образом, необходимо дальнейшее рассеивание , чтобы убедиться, что все биты ключа влияют на все биты выходного результата. Чтобы это произошло, нужно больше раундов.

Атака на двухраундовую SPN. Существует возможность расширить вышеописанные идеи, чтобы провести атаку на двухраундовую SPN с использованием подключей на каждом раунде, которая будет лучше, чем атака с применением «грубой силы»; мы оставим это здесь в качестве упражнения.

Вместо этого, мы просто отметим, что двухраундовая SPN не будет хорошей псевдослучайной перестановкой. Здесь мы опираемся на факт, упомянутый ра-

нее, о том, что лавинный эффект не происходит после всего лишь двух раундов (конечно же, это зависит от длины блока шифра и входной/выходной длины S-блоков, но с обоснованными параметрами это сработает). Атакующий сможет отличить двухраундовую SPN от универсальной перестановки, если узнает результат вычисления SPN по двум входным элементам, которые отличаются одним битом: в двухраундовой SPN много бит из двух выходных элементов будет одинаковые, что есть нечто неожиданное для случайной перестановки.)

6,2,2 Сети Фейстеля

Сети Фейстеля предлагают другой подход для построения блочных шифров. Преимущество сетей Фейстеля над постановочно-перестановочными сетями в том, что базовые функции в сети Фейстеля, в сравнении с S-блоками, используемыми в SPN, их не нужно обращать. Сеть Фейстеля, таким образом, предоставляет способ сконструировать обращаемую функцию из необращаемых компонентов. Это важно, так как хороший блочный шифр должен обладать «неструктурным» поведением (поэтому он выглядит случайным), но при этом требуя, чтобы все обращаемые компоненты конструкции по сути представляли из себя структуру. Требование обращаемости также накладывает дополнительное ограничение на S-блоки , что затрудняет их разработку.

Сеть Фейстеля работает в серии раундов. В каждом раунде ключевая раундная функция применяется, способом, описанным ниже . Раундным функциям не обязательно быть обращаемыми. Они будут ,как правило, построены из компонентов наподобие S-блоков и смешивающих перестановок, но сеть Фейстеля может работать с любыми раундными функциями, независимо от их конструкции.

В сбалансированной сети Фейстеля (единственный тип, который мы будем рассматривать) раундная функция i (f^i) принимает в качестве входного элемента подключ ki и A/2-битную строку, а выводит A/2-битную строку. Как и в случае с SPN, мастер-ключ k используется, чтобы извлекать подключи для каждого раунда. Когда какой-то мастер-ключ выбран, таким образом определив каждый подключ ki, мы определяем fi: $\{0,1\}^{A/2} \rightarrow \{0,1\}^{A/2}$ через fi(R) $\stackrel{\text{def}}{=}$ f^i(k_i, R).

Обратите внимание, что раундовые функции f^i зафиксированы и публично известны, но fi зависят от мастер-ключа и по этой причине неизвестны атакующему.

Раунд i сети Фейстеля работает следующим образом. Входной элемент раунда делится на две половины, обозначенные как L_{i-1} и R_{i-1} («левая» и «правая» половины, соответственно). Если длина блока шифра A бит, тогда L_{i-1} и R_{i-1} длиной A/2 каждая. Выходной элемент (L_i, R_i) раундов - $L_i := R_{i-1}$ и $R_i := L_{i-1} \oplus fi(R_{i-1})$. (6.3)

В r-раундовой сети Фейстеля A-битный входной элемент в сеть разбирается как (L_0, R_0), а выходной элемент является A-битным значением (L^r, R_r), полученным после применения всех r раундов. Трехраундовая сеть Фейстеля показана на Рисунке 6.5.

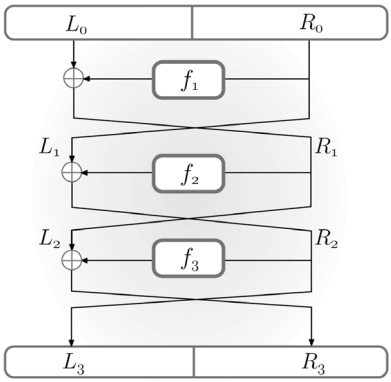

РИСУНОК 6.5: Трехраундовая сеть Фейстеля.

Обращение сети Фейстеля. Сеть Фейстеля является обращаемой, независимо от $\{f_i\}$ (и, таким образом, независимо от раундовых функций $\{f^{\wedge}_i\}$). Чтобы показать это, нам потребуется только продемонстрировать, что каждый раунд сети может быть обращен, если значение $\{fi\}$ известно. Имея выходной элемент (L_i, R_i) раунда i, мы можем высчитать (L_{i-1}, R_{i-1}) следующим образом: сначала зададим $R_{i-1} := L_i$. Затем вычислим

$L_{i-1} := R_i \oplus f_i(R_{i-1})$.

Это дает значение (L_{i-1}, R_{i-1}), которое было входным элементом этого раунда (то есть так высчитывается инверсия Уравнения (6.3)). Обратите внимание, что fi рассчитывается только в направлении вперед, потому ей необязательно быть обращаемой. Таким образом, мы имеем:

ПРЕДПОЛОЖЕНИЕ 6.4 *Пусть F будет ключевой функцией, определенной сетью Фейстеля. Тогда независимо от раундовых функций $\{f^{\wedge}_i\}$ и количества раундов, F_k является эффективно обращаемой перестановкой для всех k.*

Как и в случае с подстановочно-перестановочной сетью, атаки на сеть Фейстеля возможны, если количество раундов слишком мало. Мы увидим такие атаки, когда будем проходить DES в следующем разделе. Теоретические результаты касательно защиты сетей Фейстеля описаны в Разделе 7.6.

6.2.3DES – Data Encryption Standard

Data Encryption Standard или DES был разработан в 1970-х компание й IBM (при участии Национального агентства безопасности (National Security Agency)) и утвержден в 1977 как Федеральный стандарт по обработке информации США (Federal Information Processing Standard). В своей базовой форме DES больше не считается защищенным из-за короткой длины ключа в 56 бит, что делает его уязвимым к атакам с применением «грубой силы». Тем не менее, он все еще широко

используется сегодня в усиленной форме тройной DES, описанной в Разделе 6.2.4.

DES обладает значительной исторической важностью. Он прошел интенсивную проверку безопасности внутри криптографического сообщества, возможно даже как ни один другой криптографический алгоритм в истории. Все сходятся, как правило, в том, что, несмотря на длину ключа, DES является чрезвычайно грамотно разработанным шифром. В действительности, даже после многих лет, наиболее известная атака на DES на практике - это полный перебор всех 256 возможных ключей. (Как мы увидим, существуют важные теоретические атаки на DES, которые требуют меньше вычислений; однако, эти атаки подразумевают определенные условия, которые, кажется, тяжело реализовать на практике.)

В этом разделе мы предоставим высокоуровневый обзор главных компонентов DES. Мы настаиваем, что мы не предоставляем полную спецификацию, правильную в каждой детали, и некоторые части конструкции будут опущены в нашем описании. Нашей целью является представить базовые идеи, заложенные в конструкцию DES, но не низкоуровневые детали; читатель, который интересуется такими деталями может обратиться к ссылкам в конце данной главы.

Конструкция DES.

Блочный шифр DES - это 16-раундовая сеть Фейстеля с длиной блока в 64 бита и длиной ключа в 56 бит. Одна и та же раундовая функция f^ используется на каждом из 16 кругов. Раундовая функция принимает 46-битный подключ и, как ожидаемо для (сбалансированной) сети Фейстеля, 32-битный входной элемент (а именно, половину блока). Расписание смены ключей DES используется, чтобы извлекать последовательность из 48-битных подключей k1, . . . , k16 из 56-битного мастер-ключа. Расписание смены ключей DES относительно простое с каждым подключом ki , являющимся переставляемым подмножеством из 46 бит мастер-ключа. Для целей книги полезно будет отметить, что 56 бит мастер-ключа разделены на две половины, «левую половину» и «правую половину», каждая из которых содержит 28 бит. (Такое разделение случается после того, как начальная перестановка применяется к ключу, но мы проигнорируем это в нашем описании.) В каждом раунде самые левые 24 бита подключа принимаются как некое подмножество из 28 бит в левой стороне мастер-ключа, самые правые 24 бита раундового подключа принимаются как некое подмножество из 28 бит в правой половине мастер-ключа. Мы настаиваем, что полное расписание смены ключей (включая способ, при помощи которого биты делятся на левую и правую половины, и какие биты используются при формировании каждого подключа ki) зафиксировано и находится в публичном доступе, и единственным секретом остается сам мастер-ключ.

Раундовая функция DES Раундовая функция DES f^, иногда называемая DES mangler function, сконструирована с использованием парадигмы, которую мы анализировали

ранее: это (по сути) всего лишь подстановочно-перестановочная сеть! Если говорить более подробно, вычисление $f(k_i, R)$ с $ki \in \{0, 1\}^{48}$ и $R \in \{0, 1\}^{32}$ происходит следующим образом: сначала, R расширяется до 48-битного значения Rr. Это выполняется простой дубликацией половины битов R; мы обозначим это как $R^r := E(R)$, где

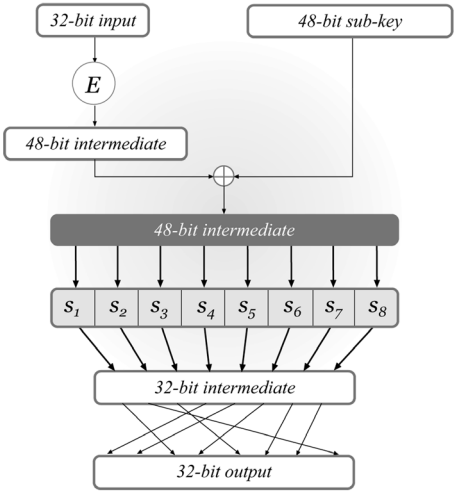

РИСУНОК 6.6: DES mangler function

E называется функция расширения. Исходя из этого, вычисления проходят точь в точь как в нашем более раннем описании SPN: Расширенное значение R^r складывается по модулю 2 с k_i, который также длиной 48 бит, и итоговое значение делиться на 8 блоков, каждый из которых длиной 6 бит. Каждый блок пропускается через (другой) S- блок, который принимает 6-битный элемент и выдает 4-битный; объединение выходных элементов из 8 S-блоков дает 32-битный результат. Затем к битам этого результата применяется смешивающая перестановка, чтобы получить конечный выходной элемент. См. Рисунок 6.6.

Одно отличие в сравнении с нашим оригинальным описанием SPN заключается в том, что S-блоки здесь не обращаемые; в действительности, они и не могут быть обращаемые, так как их входные данные длинее, чем их выходные данные. Дальнейшее обсуждение касательно структурных деталей S-блоков приведено ниже.

Мы настаиваем еще раз, что все вышеописанное (включая непосредственно S-блоки, а также смешивающая перестановка) является публичной информацией. Единственным секретом является мастер-ключ, который используется для извлечения всех подключей.

S-блоки и смешивающая перестановка. Восемь S-блоков, которые форми-

руют «ядро» f^ являются ключевым элементом конструкции DES и были очень тщательно разработаны. Изучения DES показали, что, если бы S-блоки были немного изменены, DES был бы намного более уязвим к атакам.

Это должно послужить предупреждением всем, кто желает разрабатывать блочные шифры: кажется, что произвольный выбор вовсе не произвольный, и, если он не сделан правильно, это может сделать незащищенной всю конструкцию.

Вспомним, что каждый S-блок преобразовывает 6-битный входной элемент в 4-битный выходной. Каждый S-блок может рассматриваться как таблица с 4 строками и 16 колонками, где каждая ячейка содержит 4-битную запись. 6-битный входной элемент может рассматриваться как индексацию одной из $2^6 = 64 = 4 \times 16$ ячеек таблицы следующим образом: Первый и последний бит входного элемента используется для выбора строки таблицы, а биты со 2 по 5 используются для выбора колонки таблицы. 4-битная запись на какой-то из позиций таблицы представляет собой выходное значение для входного элемента, связанного с этой позицией.

S-блоки DES имеют следующие свойства (помимо прочего):

1. Каждый S-блок - это функция 4-в-1. (То есть ровно 4 входных элемента преобразовываются в каждый возможный выходной элемент.) Это проистекает из свойства ниже.

2. Каждая строка таблицы содержит каждый из 16 возможных 4-битных строк ровно один раз.

3.Изменение одного бита любого входного элемента в S-блок всегда меняет как минимум два бита выходного элемента.

Смешивающая перестановка всегда разрабатывалась с осторожностью. В частности, она имеет такое свойство, что четыре выходных бита из любого S-блока будут влиять на входной элемент в 6 S-блоков в следующем раунде. (Это возможно из-за функции расширения, которая применяется в следующем раунде перед вычислением S-блоков.)

Лавинный эффект DES. Конструкция «mangler function» гарантирует, что DES представляет сильный лавинный эффект. Для того, чтобы убедиться в этом, мы проследим за разницей между промежуточными значениями в вычислении DES двух входных элементов, которые отличаются всего одним битом. Давайте обозначим два входных элемента шифра как (L0, R0) и (L^r , R^r), где мы предположим, что $R_0 = R^r$, и поэтому будет иметь место разница в один бит в левой половине элементов (возможно, в дальнейшем поможет обращение к Уравнению (6.3) и Рисунку 6.6). После первого раунда промежуточные значения (L_1, R_1) и (L^r , R^r) все еще отличаются на один бит, хотя теперь это отличие находится в правой половине. Во втором раунде DES правая половина каждого из входных элементов проходит через f^. Учитывая, что бит, которым отличаются R_1 и R^r не дублиро-

вался при расширении, промежуточные значения перед применением S-блоков все еще отличаются всего лишь на один бит. Согласно свойству 3 S-блоков, промежуточные значения после вычисления S-блока отичаются как минимум на два бита. В результате промежуточные значения (L_2, R_2) и (L^r, R^r) отличаются на три бита: 1-битная разница присутствует между L_2 и L^r (перенесенная из разницы между R_1 и R^r) и 2-битная разница между R_2 и R^r.

Смешивающая перестановка распространяет двухбитную разницу между R_2 и R^r так, что в следующем раунде каждый из двух бит используется в качестве входного элемента в другой S-блок, из чего получается разница в как минимум 4 бита в правых половинах промежуточных значений. (Если один или оба из двух бит, которыми отличаются R_2 и R^r, продублируются функцией E, разница может быть еще большей.) Теперь еще и в левых половинах присутствует 2-битная разница. Как и в случае с подстановочно-перестановочной сетью, мы имеем экспоненциальный эффект, и поэтому после 7 раундов мы прогнозируем, что в правой половине затронуты будут все 32 бита (а после 8 раундов будут затронуты все 32 бита и в левой половине).

В DES 16 раундов, и поэтому лавинный эффект при вычислении случается очень рано. Таким образом обеспечиваются такие вычисления в DES, при которых одинаковые входные данные дают выходные данные, похожие на независимые.

Атаки на DES с уменьшиным числом раундов

Полезным упражнением для более углубленного понимания конструкции DES и ее безопасности будет наблюдение за поведением DES с всего несколькими раундами. Мы покажем атаки на одно-, двух-, и трехраундовые варианты DES (напоминаем, что настоящий DES имеет 16 раундов). DES с тремя раундами или менее не может быть псевдослучайной функцией, потому что три раунда не достаточно для того, чтобы произошел лавинный эффект. Таким образом, мы заинтересованы в демонстрации более трудных (и вредящих) атак с восстановлением ключа, которые высчитывают ключ k, используя относительно маленькое количество входных/выходных пар, высчитаных с использованием ключа. Некоторые из атак похожи на те, что мы наблюдали в контексте подстановочных-перестановочных сетей; здесь, однако, мы увидим, как они применяются к конкретному блочному шифру, а не к какой-то абстрактной конструкции.

Атаки ниже, будут атаками на основе открытого текста, в которых злоумышленник знает какие-то пары открытого текста/шифротекста $\{(x_i, y_i)\}$ при yi = DESk (xi) для какого-то секретного ключа k. При описании атак мы будем акцентировать внимание на определенной входной/выходной паре (x, y) и предоставим информацию о ключе, который злоумышленник может извлечь из этой пары. Продолжая использовать обозначения, разработанные ранее, мы обозначим левую и правую половины входного элемента x как L0 и R0, соответсвенно,

и пусть Li, Ri обозначают левую и правую половины после раунда i. Вспомним, что E обозначает функцию расширения DES, ki обозначает подключ, используемый в раунде i, и fi(R) = f(ki, R) обозначает собственно функцию, применяемую в сети Фейстеля в раунде i.

Одноразундовый DES. Скажем, у нас имеется входная/выходная пара (x, y). В одноразундовом DES y = (L_1, R_1), где $L_1 = R_0$ и $R_1 = L_0 \oplus f1(R0)$. Мы, таким образом, знаем входную/выходную пару для f1; в частности, мы знаем, что $f_1(R_0) = R_1 \oplus L_0$. Путем применения обращения смешивающей перестановки к выходному элементу $R_1 \oplus L_0$ мы получим промежуточное значение, состоящее из выходных данных со всех S-блоков, где первые 4 бита - это выходной элемент из первого S-блока, следующие 4 бита - выходной элемент из второго S-блока и так далее.

Рассмотрим (известный) 4-битный выходной элемент из первого S-блока. Поскольку каждый S-блок является функцией 4-к-1, это означает, что существуют ровно четыре возможных входных элемента в этот S-блок, которые превратятся в имеющийся выходной элемент, то же самое и с другими S-блоками; каждый такой входной элемент длиной 6 бит. Входной элемент в S-блоки - это просто XOR $E(R_0)$ с подключом k1. Поскольку R0 и, следовательно, E(R0) нам известен, мы можем вычислить набор из четырех воможных значений для каждой 6-битной части k1. Это значит, что мы уменьшили число возможных ключей k_1 с 2^{48} до $4^{48/6} = 4^8 = 2^{16}$ (поскольку существуют 4 варианта для каждой из восьми 6-битных частей k_1). Это уже небольшое число, и поэтому мы можем просто попробовать все варианты на другой входной/выходной паре (xr , yr) найти правильный ключ. Таким образом, мы получим ключ, используя только два известных открытых текста за время приблизительно 2^{16}.

Двухраундовый DES. В двухраундовом DES, выходной элемент y равен (L2, R2), где

$L_1 = R_0$

$R_1 = L_0 \oplus f_1(R_0)$

$L_2 = R_1 = L_0 \oplus f_1(R_0)$

$R_2 = L_1 \oplus f_2(R_1)$.

L_0, R_0, L_2 и R_2 известны из полученной входной/выходной пары (x, y), и, таким образом, мы также знаем $L_1 = R_0$ и $R_1 = L_2$. Это означает, что мы знаем вхоной/выходной элементы и f_1, и f_2, и поэтому тот же самый метод , который был использован для атаки на одноразундовый DES, может быть использован здесь для определения как k_1, так и k2 за время приблизительно 2 • 216. Данная атака работает, даже если k_1 и k_2 являются полностью независимыми ключами, хотя, фактически, расписание смены ключей DES подтверждает, что многие из битов k_1 и k_2 равны (что можно использовать для ускорения атаки в дальнейшем).

Трехраундовый DES. Согласно Рисуноку 6.5, выходное значение у теперь равняется (L_3, R_3). Поскольку $L_1 = R_0$ и $R_2 = L_3$, есдинственными неизвестными значениями на рисунке остаются R_1 и L_2 (которые равны).

Теперь у нас уже больше нет входной/выходной пары для любой раундовой функции fi. Например, выходное значение f_2 равняется $L_1 \oplus R_2$, где оба этих значения равны. Однако, мы не знаем значения R_1, которое является входным элементом f_2. Аналогично, мы можем определить входные элементы f_1 и f_3 , но не выходные элементы этих функций. Таким образом, атака, которую мы использовали, чтобы взломать одноразундовый DES, здесь не сработает.

Вместо того, чтобы надеяться на знания о входных и выходных элементах одной из раундовых функций, мы используем знания об определенном отношении между входными и выходными данными f_1 и f_3. Обратите внимание, что выходной элемент f_1 равен $L_0 \oplus R_1 = L_0 \oplus L_2$, а выходной элемент f_3 равен $L_2 \oplus R_3$. Следовательно, $f_1(R_0) \oplus f_3(R_2) = (L_0 \oplus L_2) \oplus (L_2 \oplus R_3) = L_0 \oplus R_3$, где известны и L_0, и R_3. То есть XOR выходных данных f_1 и f_3 известен. Более того, входной элемент в f_1 - это R_0 , а входной элемент в f_3 - это L_3, оба они известны. Подытоживая, можно сказать, что мы можем определить входные данные в f_1 и f_3, а также XOR их выходных данных. Сейчас мы опишем атаку, которая находит секретный ключ на основе данной информации.

Вспомним, что расписание смены ключей DES имеет такое свойство, что мастер-ключ делится на «левую половину», которую мы обозначили как kL, и «правую половину» kR, каждая из которых содержит 28 бит. Более того, 24 самых левых бита подключа, используемых в каждом раунде берутся только из kL, а 24 самых правых бита каждого подключа берутся из kR. Это означает, что kL влияет толко на входные данные в первые четыре S-блока в раунде, в то время как kR влияет только на входные данные в последние четыре S-блока. Поскольку смешивающая перестановка известна, мы также знаем, какие биты выходного элемента каждой раундовой функции выходят из каждого S-блока.

Идея, что кроется в атаке, - это по-отдельности пересечь пространство ключей за каждой половиной мастер-ключа, осуществляя атаку со сложностью приблизительно $2 \cdot 2^{28}$, а не 2^{56}. Такая атака будет возможна, если мы сможем проверить догадку о половине мастер-ключа, и мы сейчас покажем, как это можно сделать. Скажем, мы угадали какое-то значение для kL, левую половину мастер-ключа. Мы знаем входной элемент R_0 функции f1, и поэтому используя угадывание kL , мы можем вычислить входной элемент в первые четыре S-блока. Это означает, что мы можем вычислить половину выходных битов f1 (смешивающая перестановка распространяет биты, которые мы знаем, но поскольку смешивающая перестановка известна, мы знаем точно, какие это биты). Аналогичным образом мы можем вычислить одинаковые расположения в выходном элементу f3 , используя известный входной элемент L3 в f3 , а также ту же догадку для kL.

Наконец, мы можем вычислить XOR этих выходных значений и проверить, совпадают ли соответствующие биты, в известном значении XOR выходных данных f_1 и f_3. Если они не равны, тогда наша догадка для kL является некорректной . Корректная догадка для kL будет всегда проходит этот тест, и поэтому не будет исключена, но некорректная догадка ожидаемо пройдет этот тест только с вероятностью приблизительно 2^{-16} (поскольку мы проверяем равенство 16 бит в двух вычисленных значениях). Существует 2^{28} возможных значений для kL, поэтому если каждое некорректное значение остается жизнеспособным возможным значением с вероятностью2^{-16} , тогда мы ожидаемо останемся с только лишь $2^{28} \bullet 2^{-16} = 2^{12}$ вариантов для kL после вышеописанного.

Выполняя вышеописанное для каждой половины мастер-ключа, мы получим за время $2 \bullet 2^{28}$ приблизительно 2^{12} возможных значений для левой половины и 2^{12} возможных значений для правой половины. Поскольку каждая комбинация левой и правой половин возможна, у нас есть 224 возможных ключей, и мы можем запустить перебор с использованием «грубой силы» по этому множеству, используя дополнительную входную/выходную пару (xr , yr). (Альтернативный подход, который более эффективный, - это просто повторить предыдущую атаку, используя 2^{12} оставшихся возможных значений для каждой половины ключа.) Временная сложность атаки составляет приблизительно $2 \bullet 2^{28} + 2^{24} < 2^{30}$, что намного меньше, чем 2^{56}.

Безопасность DES

После почти 30 лет интенсивного изучения наилучшей известной практической атаки , на DES все еще остается полный перебор его пространства ключей. (Мы обсудим кое-какие важные теоретические атаки в Разделе 6.2.6 Эти атаки требуют большого количества входных/выходных пар, которые было бы тяжело получить для атаки на настоящую систему, которая использует DES.) К несчастью, 56-битная длина ключа DES достаточно коротка, чтобы полный перебор всех 256 вариантов ключей был осуществим в наше время. Уже в конце 1970-х существовали сильные возражения касательно выбора такого короткого ключа для DES. В те времена возражение было теоретическим, так как вычислительная мощность, необходимая, чтобы перебрать такое количество ключей была, в общем, недоступна. 4 Целесообразность атаки с применением «грубой силы» на DES, однако, была продемонстрирована в 1997, когда первый набор испытаний DES, установленный RSA Security, был пройден группой DESCHALL Project с использованием нескольких тысяч компьютеров координированных по интернету; вычисление заняло 96 дней. Второе испытание было пройдено в следующем году всего за 41 день участниками проекта distributed.net. Серьезный прорыв случился в 1998, когда третье испытание было пройдено всего за 56 часов. Этот впечатляющий подвиг был осуществлен посредством узкоспециализированной машины для взлома DES под названием Deep Crack , которая была построена Electronic Frontier Foundation и стоила 250000 дол-

ларов. В 1999 году испытание DES было пройдено всего за 22 часа совместными усилиями Deep Crack и distributed.net. Сегодня самым ультрасовременным взломывателем DES является DES cracking box от компании PICO Computing, который использует 48 FPGA и может найти ключ DES за приблизительно 23 часа.

Компромиссы пространства/времени, описанные в Разделе 5.4.3 показывают, что атаки с полным перебором могут быть ускорены предварительной обработкой и дополнительной памятью. Из-за короткой длины ключа DES компромиссы пространства/времени могут быть особенно эффективны. В частности, используя предварительную обработку, можно генерировать таблицу размером в несколько терабайт, которая затем сможет извлечь ключ DES с высокой вероятностью из одной входной/выходной пары, использовав приблизительно 238 вычислений DES (на это могут уйти буквально минуты). Нижняя грань - это то, что DES имеет ключ, который очень и очень короток и не может сегодня считаться защищенным для любого серьезного применения.

Вторая причина для беспокойства - это относительно короткая длина блока DES. Короткая длина блока является проблемой, потому что конкретная защита многих конструкций на основе блочного шифра зависит от длины блока, даже если используемый шифр является «безупречным.» Например, доказательство защиты для режима CTR (сравните С Теоремой 3.32) показывает, что даже когда полностью случайная функция используется, атакующий может взломать защиту такой схемы шифрования с вероятностью $2q^2/2^A$, если он получит q пар открытого текста/шифротекста. В случае с DES, где $A = 64$, это означает, что, если атакующий получит только лишь $q = 2^{30}$ пар открытого текста/шифротекста, защита будет взломана с высокой вероятностью. Получить пары открытого текста/шифротекста довольно легко, если злоумышленник перехватывает шифрование сообщений , содержащее известные заголовки, избыточности и пр.

Незащищенность DES не имеет по сути ничего общего со своей конструкцией, но все дело в его короткой длине ключа (и в меньшей степени, короткой длине блока). Нужно отдать должное разработчикам DES, которые, казалось, смогли создать почти безупречный блочный шифр (несмотря на его слишком короткий ключ). Так как DES сама по себе не имеет существенных структурных слабостей, есть смысл использовать DES в качестве структурного элемента при создании блочных шифров с более длинными ключами. Мы рассмотрим это позже в Разделе 6.2.4.

Замена DES, Advanced Encryption Standard (AES), о котором речь пойдет далее в этой главе, был четко разработан с учетом проблем, касающихся короткой длины ключа и короткой дляны блока DES. AES поддерживает 128-, 192- или 256-битные ключи, а также длину блока 128 бит.

---

[4]В 1977 году компьютер, который мог бы взломать DES за один день, был оценен в 20 миллионов долларов.

Атаки, которые были более продвинуты, чем применение «грубой силы» к DES впервые были продемонстрированы в начале 1990-х Бигамом (Biham) и Шамиром (Shamir), которые разработали метод под названием дифференциальный криптоанализ. Их атака продолжалась по времени 237 и требовала 247 подобранных открыты текстов. В то время, как атака была прорывом с теоретической точки зрения, на практике же она не мела никакого смысла, поскольку трудно представить себе реалистичный сценарий, когда злоумышленник может получить настолько много шифрований подобранного открытого текста.

Интересно отметить, что работа Бигама и Шамира указала на то, что S-блоки DES были разработаны исключительно для того, чтобы выдерживать дифференциальный криптоанализ, как будто дифференциальный криптоанализ был известен разработчиками DES (хотя они в этом не признавались). После того, как Бигам и Шамир обнародовали свои результаты, подозрения были подтверждены.

Линейный криптоанализ был разработан Мацуи (Matsui) в середине 90-х годов и также успешно применялся к DES. Преимущество такой атаки в том, чтобы скорее использовать известные открытые тексты, чем подобранные открытые тексты. Тем не менее, количество необходимых пар открытого текста/шифртекста, около 243, все еще велико.

Мы кратко описали дифференциальный и линейный криптоанализы в Разделе 6.2.6.

6.2.4 DES: Увеличение длины ключа блочного шифра

Главная слабость DES - короткий ключ. Таким образом, имеет смысл попробовать разработать блочный шифр с более длинным ключом, используя DES в качестве структурного элемента. Некоторые методы выполнения этой задачи обсуждаются в данном разделе. Хотя в процессе обсуждения мы часто ссылаемся на DES (Data Encryption Standard - Стандарт шифрования данных), который является наиболее известным блочным шифром, к которому были применены эти методы, все, что мы говорим здесь, относится к любому блочному шифру.

Внутренние модификации в зависимости от конструкций «черного ящика». Имеется два метода, один из которых можно было бы принять для построения другого шифра на основе DES. Первый метод состоит в том, чтобы каким-либо образом изменить внутреннюю структуру DES при увеличении длины ключа. Например, можно было оставить нетронутой раундовую функцию и просто использовать 128-битный мастер-ключ с другим ключевым заданием (по-прежнему выбрав вспомогательный 48-битный ключ в каждом раунде. Или можно изменить сами S-блоки и использовать вспомогательный ключ больше

---

[5] Поэтому изменение внутреннего содержания блочного шифра не рекомендуется.

го размера в каждом раунде. Недостаток таких методов заключается в том, что при модификации DES, даже минимальным образом, теряется достоверность в силу того, что он остается устойчивым к атакам в течение стольких лет. Криптографические конструкции очень чувствительны, и даже небольшие, на вид, незначительные изменения могут сделать конструкцию полностью небезопасной.

Альтернативный метод, который не затрагивает указанной выше проблемы заключается в том, чтобы использовать DES в качестве черного ящика и совсем не затрагивать внутреннюю структуру. При таком методе DES мы рассматриваем в качестве «совершенного» блочного фильтра с 56-битным ключом и конструируем новый блочный шифр, который только использует исходный немодифицированный DES. Поскольку сам по себе DES не затрагивается, то это гораздо более разумный метод и это - тот метод, который будет рассматриваться здесь.

Двукратное шифрование

Пусть F будет блочным шифром с ключом длиной n- битов и длиной блока A-битов.

Тогда новый блочный шифр F r с ключом длиной 2n может быть определен согласно

$$F'_{k_1,k_2}(x) \stackrel{\text{def}}{=} F_{k_2}(F_{k_1}(x)),$$

где k1 and k2 – это независимые ключи. Для случая, когда F – это DES, мы получим шифр F r , называемый 2DES, для которого требуется 112-битный ключ, если полный поиск ключа был бы наилучшей доступной атакой, то ключ длиной 112 битов был бы достаточным, поскольку атака, требующая времени 2112 совершенно невозможна. К сожалению, мы сейчас выявим атаку на F r , которая проходит в течение приблизительно 2n, что существенно меньше времени 2^{2n}, которое, как можно было бы надеяться, необходимо для полного поиска для 2n-битного ключа. Это означает, что новый блочный шифр, по существу, не лучше старого, даже несмотря на то, что у него есть ключ, который в два раза длиннее.6

Эта атака называется «атакой встречи посередине» по причинам, которые скоро станут понятны. Скажем, оппонент получает отдельную входную/выходную пару (x, y), где y = F r* * (x) = Fk* (Fk* (x)) для неизвестного k* , k*. Оппонент может сузить множество возможных ключей следующим образом:

1. Для каждого k1 ∈ $\{0, 1\}^n$, вычисляем z := Fk (x) и сохраняем (z, k1) в списке L.
2. Для каждого k2 ∈ $\{0, 1\}^n$, вычисляем z := F −1(y) и сохраняем (z, k2) в списке Lr.

6Это не совсем верно, поскольку атака простым последовательным перебором на F может быть осуществлена за время 2n и с постоянным объемом памяти, тогда как атака на F r , которую мы показываем, требует 2n времени и 2n памяти. Тем не менее, эта атака показывает, что F r не достигает желаемого уровня безопасности.

3. Записи (z1, k1) ∈ L и (z2, k2) ∈ L^r совпадают, если z1 = z2. Для каждого такого совпадения добавьте (k1, k2) в множество S. (Совпадения можно легко обнаружить после сортировки L и Lr по их первым компонентам).

Графическое описание атаки см. на рис. 6.7.

Эта атака занимает времени $O(n \cdot 2^n)$ и требует объема памяти O((n + A) • 2n). Вывод множества S по этому алгоритму содержит в точности те же значения (k1, k2), для которых

$$F_{k_1}(x) = F_{k_2}^{-1}(y) \qquad (6.4)$$

или, что то же самое, для которых y = F r (x). В частности, (k* , k*) ∈ S. С другой стороны, ожидается (эвристически), что пара (k1, k2) *f*= (k* , k*) будет удовлетворять Выражение (6.4) с вероятностью 2−A , если рассматривать Fk (x) и F −1(y) как равномерные

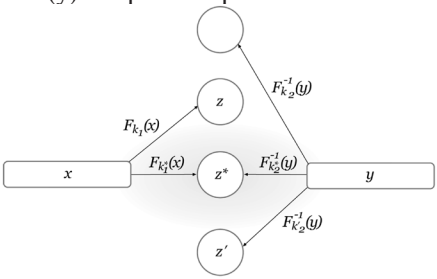

РИС. 6.7: Атака встречи на середине.

Строки A-битов и, таким образом, ожидаемый размер S составляет 22n • 2−A = 22n−A . Используя несколько других входных/выходных пар и выполнив пересечение множеств, которые получены, правильное (k* , k*) можно идентифицировать с очень высокой вероятностью.

Тройное шифрование

Очевидное обобщение предыдущего подхода состоит в том, чтобы применить блочный шифр три раза подряд. Общими являются два варианта такого метода:

Вариант 1: три ключа. Выберите три независимых ключа k1, k2, k3 и определите

$$F''_{k_1,k_2,k_3}(x) \stackrel{\text{def}}{=} F_{k_3}(F_{k_2}^{-1}(F_{k_1}(x))).$$

Вариант 2: два ключа. Выберите два независимых ключа k1, k2 и затем определите

$$F''_{k_1,k_2}(x) \stackrel{\text{def}}{=} F_{k_1}(F_{k_2}^{-1}(F_{k_1}(x)))$$

Прежде чем сравнивать безопасность этих двух альтернативных вариантов,

заметим, что восстанавливается среднее инициирование F . Если F – достаточно хороший шифр, то это не имеет никакого значения по отношению к безопасности, поскольку если F представляет собой строгую псевдослучайную перестановку, то также должно быть F^{-1} . Причина для восстановления второго применения F заключается в том, чтобы получить обратную совместимость: если задать k1 = k2 = k3, то результирующая функция эквивалентна одному инициированию F с использованием ключа k1.

Безопасность первого варианта. Длина ключа первого варианта равна 3n и, таким образом, можно надеяться, что лучшая атака на этот шифр потребует времени 23n. Однако этот шифр подвержен атаке «встречи посередине» так же, как в случае двойного шифрования, хотя атака требует времени 22n. Это самая известная атака. Таким образом, хотя этот вариант не так безопасен, как можно было бы надеяться, он приобретает достаточную безопасность для всех практических целей, даже для n = 56 (при условии, конечно, исходный шифр F не имеет недостатков).

Безопасность второго варианта. Длина этого ключа составляет 2n и, таким образом, лучшее, на что можно надеяться - это безопасность по отношению к атакам, действующим в течение времени 22n. Не известна атака с лучшей временной сложностью, когда оппоненту известно лишь небольшое количество входных/выходных пар. (См. Пример 6.13 для атаки, использующей 2n выбранных нешифрованных текстов). Таким образом, тройное шифрование с двумя ключами - это разумный практический выбор.

Тройной-DES (3DES). Тройной-DES (или 3DES) основывается на тройном инициировании DES с использованием двух или трех ключей, как описано выше. 3DES был стандартизован в 1999 г. и широко используется в настоящее время. Основными его недостатками являются относительно небольшая длина блока и тот факт, что он работает сравнительно медленно, поскольку требуются 3 полных операции блочного шифрования. Поскольку в настоящее время минимальная рекомендованная длина ключа составляет 128 битов, то 3DES с двумя ключами более не рекомендуется (потому, что его длина ключа равна только 112 битам). Эти недостатки привели к замене DES/тройного DES Расширенным Стандартом Шифрования (Advanced Encryption Standard), представленным в следующем разделе.

6.2.5 AES – Расширенный Стандарт Шифрования

В январе 1997 г. Институт Стандартов и Технологии (NIST) США объявил, что проведет конкурс, чтобы выбрать новый блочный шифр, который будет называться Расширенным Стандартом Шифрования, или AES, чтобы заменить DES. Конкурс начался с открытого вызова участников, чтобы представить варианты блочных шифров для оценки. Были представлены на рассмотрение в общей сложности 15 различных алгоритмов, в том числе статьи многих лучших

криптографов и криптоаналитиков. Каждый вариант шифра команды тщательно анализировали члены NIST, публика и (в особенности) другие команды. Были проведены два семинара, по одному в 1998 г. и в 1999 г., чтобы обсудить и проанализировать различные представленные варианты. После второго семинара NIST сузил круг претендентов до 5 финалистов и начался второй этап конкурса. В апреле 2000 г. состоялся третий семинар AES, на который были приглашены для дополнительного изучения пять финалистов. В октябре 2000 г. NIST объявил, что победил алгоритм Rijndael (Рийндаель - блочный шифр, разработанный бельгийскими криптографами Винсентом Рийменом (Vincent Rijmen) и Джоан Даемен (Joan Daemen)), хотя NIST признал, что алгоритм любого из 5 финалистов был бы отличным выбором. В частности, никаких серьезных уязвимостей для безопасности не было обнаружено ни в одном из 5 финалистов, а выбор «победителя» был частично основан на таких свойствах, как эффективность, производительность на аппаратном уровне, гибкость и т.д.

Процесс выбора AES был изобретательным, поскольку любая группа, которая представила алгоритм и поэтому была заинтересована в принятии своего алгоритма, имела сильную мотивацию обнаружить атаки на другие представленные варианты. Таким образом, лучшие криптоаналитики мира сконцентрировали свое внимание на поиске даже малейших недостатков шифров, представленных на конкурс. Всего лишь через несколько лет каждый вариант алгоритма был подвергнут интенсивному изучению, увеличив, таким образом, нашу уверенность в безопасности победившего алгоритма. Безусловно, чем дольше алгоритм используется и изучается без нарушения, тем больше будет продолжать расти наша уверенность в нем. В настоящее время AES широко используется и никаких существенных недостатков безопасности обнаружено не было.

Структура AES. В этом разделе мы представляем высокоуровневую структуру Рийндаель/AES. (Говоря техническим языком, Рийндаель и AES - это не одно и то же, но различия несущественны для нашего обсуждения здесь). Как и в случае DES, мы не представляем полную спецификацию и наше описание не следует использовать в качестве основы для реализации. Наша цель состоит в том, чтобы дать только общее представление о том, как работает алгоритм.

Блочный шифр AES имеет длину блока 128 бит и может использовать 128, 192 и 256-битные ключи. Длина ключа влияет на список ключей (то есть, на вспомогательный ключ, который используется в каждом раунде), а также количество раундов, но не влияет на структуру высокого уровня каждого раунда.

В отличие от DES, который использует структуру Фейстеля, AES, по существу, является сетью замены-перестановки. Во время вычисления алгоритма AES массив байтов 4 на 4, называемый состоянием, модифицируется в серии раундов. Изначально состояние устанавливается равным входу в шифр (обратите внимание, что вход составляет 128 битов, которые равны точно 16 байтам).

Затем к состоянию применяются следующие операции в серии четырех этапов во время каждого раунда:

Этап 1 – Добавить ключ раунда: В каждом раунде AES, 128-битный вспомогательный ключ получается из главного ключа, и интерпретируется как массив байтов 4 на 4. Массив состояния обновляется посредством операции исключающего ИЛИ с помощью этого вспомогательного ключа.

Этап 2 – Вспомогательные байты: На этом этапе каждый байт массива состояния заменяется другим байтом в соответствии с единой фиксированной таблицей перекодировки S. Эта таблица замены (или S-блок) является взаимнооднозначным соответствием по {0, 1}8.

Этап 3 – Сдвиг строк: На этом этапе байты в каждой строке массива сдвигаются влево следующим образом: первая строка массива остается нетронутой, вторая строка сдвигается на одну позицию влево, третья строка сдвигается на две позиции влево и четвертая строка сдвигается на три позиции влево. Все сдвиги - циклические, например, во второй строке первый байт становится четвертым байтом.

Этап 4 – Перемешивание столбцов: На этом этапе применяется обратимое преобразование к четырем байтам в каждом столбце. (С технической точки зрения это линейное преобразование, т. е. матричное умножение над соответствующим полем). Это преобразование обладает таким свойством, что если два входа отличаются в $b > 0$ байтах, то применение преобразования дает два выхода, отличающихся, как минимум, в $5 - b$ байтах.

В конечном раунде Перемешивание столбцов заменяется Добавить ключ раунда. Это препятствует оппоненту просто инвертировать последние три этапа, которые не зависят от ключа.

Рассматривая вместе этапы 3 и 4 в качестве этапа «смешивания», мы видим, что каждый раунд AES имеет структуру сети замены-перестановки: вспомогательный ключ раунда сначала обрабатывается исключающим ИЛИ вместе с входом в текущий раунд; далее небольшая обратимая функция применяется к «порциям данных» результирующего значения; наконец, биты результата смешиваются, чтобы получить диффузию (преобразование открытого текста для нарушения его статистической структуры). Единственная разница состоит в том, что в отличие от нашего предыдущего описания сетей замены-перестановки, в данном случае этап смешивания не состоит из простой перетасовки битов, а вместо этого проводится перестановка плюс обратимое линейное преобразование. (Немного упрощая дело и рассматривая тривиальный 3-битный пример, перетасовывая биты x = x1»x2»x3 можно, например, отобразить x к xr = x2»x1»x3. Обратимое линейное преобразование можно отобразить x к x1 ⊕ x2»x2 ⊕ x3»x1 ⊕ x2 ⊕ x3.)

Количество раундов зависит от длины ключа. Десять раундов используется для 128-битного ключа, 12 раундов для 192-битного ключа и 14 раундов для 256-битного ключа.

Безопасность AES. Как уже упоминалось, шифр AES подвергался тщательному изучению в процессе выбора и продолжает изучаться до сих пор. На сегодняшний день не существует никаких практических криптоаналитических атак, которые значительно лучше, чем полный поиск ключа.

Мы пришли к выводу, что по состоянию на сегодняшний день AES представляет собой прекрасный выбор для любой криптографической схемы, которая требует (строгой) псевдослучайной перестановки. Этот шифр бесплатный, стандартизованный, эффективный и с высоким уровнем безопасности.

6.2.6 *Дифференциальный и линейный криптоанализ

Блочные шифры относительно сложны и сами по себе трудны для анализа. Тем не менее, не следует никого вводить в заблуждение, полагая, что сложный шифр трудно взломать. Наоборот, очень трудно создать безопасный блочный шифр и удивительно легко найти атаки на большинство конструкций (независимо от того, насколько сложными они представляются). Это должно служить предупреждением, что неспециалисты не должны пытаться создавать новые шифры. Учитывая доступность тройного DES и AES, трудно оправдать использование чего-то еще.

В этом разделе мы опишем два инструмента, которые теперь являются стандартной частью набора инструментов криптоаналитика. В данном случае наша цель заключается в том, чтобы придать вкус некому расширенному криптоанализу, а также подкрепить идею, что разработка безопасного блочного шифра включает в себя тщательный выбор его компонентов.

Дифференциальный криптоанализ. Этот метод, который может привести к атаке выбранного открытого текста на блочном шифре, был впервые представлен в конце 1980-х гг. Бихэмом (Biham) и Шамиром (Shamir), которые использовали его для атаки DES в 1993 г. Основная идея атаки заключается в том, чтобы представить в табличной форме конкретные различия во входных данных, которые приводят к конкретным различиям в выходных данных с вероятностью большей, чем можно было бы ожидать от случайной перестановки. В частности, пусть дифференциал $(\Delta x, \Delta y)$ появляется в некой ключевой перестановке F_r с вероятностью p, если для однородных входных данных $x1$ и $x2$, удовлетворяющих $x1 \oplus x2 = \Delta x$, и однородном выборе ключа k, вероятность того, что $k(x1) \oplus Fk(x2) = \Delta y$ равна p. Для любого фиксированного $(\Delta x, \Delta y)$ и $x1, x2$, удовлетворяющих $x1 \oplus x2 = \Delta x$, если мы выбираем однородную функцию $f: \{0,1\}^A \rightarrow \{0,1\}^A$, имеем $Pr[f(x1) \oplus f(x2) = \Delta y] = 2^{-A}$. В плохом блочном шифре, тем не менее, могут быть дифференциалы, которые возникают с существенно большей вероятностью. Это

может быть использовано, чтобы обеспечить атаку с полным восстановлением ключа, как мы сейчас наблюдаем для сетей с защитой от несанкционированного доступа (SPN - subscriber-protected network).

Мы опишем основную идею, а затем проработаем ее на конкретном примере. Пусть F будет блочным шифром с A-битной длиной блока, который является r-раундовой сетью SPN и допустим, что k (x) означает промежуточный результат в расчете Fk (x) после применения этапа ключевого смешивания раунда r. (T.e., F r исключает подстановку S-блока и перемешивающей перестановки последнего раунда, а также этап итогового ключевого смешивания). Допустим, имеется дифференциал (Δx, Δy) in F r , который появляется с вероятностью p 2−A. Можно использовать этот дифференциал с высокой вероятностью, чтобы узнать биты вспомогательного ключа итогового смешивания kr+1. Идея высокого уровня следующая: пусть {(xi , xi)}L будет набором L пар случайных входных данных с дифференциалом Δx, т.е., с xi ⊕ xi = Δx для всех i. Используя атаку подобранного открытого текста, получим значения yi= Fk (xi) и yi= Fk (xi) для всех i. Теперь, для всех возможных строк битов k∈ {0, 1}A, вычислим y˜i = F r (xi) и y˜i = F r (xi), предполагая, что итогового вспомогательного ключа kr+1 равно k*. Это делается инвертированием итогового ключа смешивания с использованием k*, а затем инвертирования смешивающей перестановки и S-блоков раунда r, которые не зависят от мастер-ключа. Когда k* = kr+1, мы предполагаем, что доля p пар будет удовлетворять y˜i ⊕ y˜i = Δy . С другой стороны, когда k* *f*= kr+1 мы эвристически можем ожидать только 2−A-долю пар, чтобы получить этот дифференциал. При задании L достаточно большим можно определить правильное итоговое значение вспомогательного ключа kr+1 .

Это работает, но не очень эффективно, поскольку на каждом шаге мы перечисляем более 2A возможных значений. Можно сделать это лучше, одновременно угадав части kr+1 . Более конкретно, предположим, что S-блоки в F имеют длину ввода/вывода 1 байт, и сосредоточимся на первом байте Δy , который, как мы предполагаем, отличен от нуля. Можно проверить, содержит ли дифференциал в этом байте только 8 битов kr+1, а именно те 8 битов, которые соответствуют (после раунда r смешивающей перестановки) выходным данным первого S-блока. Таким образом, действуя как указано выше, мы можем изучать эти 8 бит путем перечисления по всем возможным значениям для этих битов, и видеть, какое значение дает искомый дифференциал в первом байте с наибольшей вероятностью. Неверные прогнозы для этих 8 битов дают ожидаемый дифференциал в этом байте с (эвристической) вероятностью 2−8, но правильный прогноз даст ожидаемый дифференциал с вероятностью примерно p +2−8; это потому, что с вероятностью

р дифференциал имеется на всем блоке (так, в частности, для первого байта) и, если это не так, то мы можем рассматривать дифференциал в первом байте как случайный. Обратите внимание, что различные дифференциалы могут потребоваться, чтобы изучить различные части kr+1.

На практике различные оптимизации выполняются, чтобы улучшить эффективность указанного выше теста, или, более конкретно, чтобы увеличить разрыв между вероятностью того, что неправильный прогноз дает дифференциал в зависимости от вероятности того, что дает правильный прогноз. Одна оптимизация заключается в использовании дифференциала с низким весом, в которой Δу имеет много нулевых байтов в позициях, которые входят в S-блоки в раунде r. Любые пары у˜1, у˜2, удовлетворяющие такой дифференциал, имеют равные значения, входящие в множество S-блоков в раунде r, и это приведет к выходным значениям у1, у2, которые равны в соответствующих позициях битов (в зависимости от заключительной смешивающей перестановки). Это означает, что при выполнении теста, описанного ранее, можно просто отказаться от любых пар (yi , yi), которые не согласуются в этих позициях битов (поскольку соответствующие промежуточные значения (у˜1, у˜2), вероятно, не могут удовлетворить дифференциал для любого выбора итогового вспомогательного ключа). Это значительно повышает эффективность атаки.

Если kr+1 известно, то взломщик может отменить заключительный этап смешивания ключа, а также смешивающую перестановку и этапы подстановки S-блока раунда r (поскольку они не зависят от мастер-ключа), а затем применить ту же атаку, используя отличающийся дифференциал, чтобы найти вспомогательный ключ kr r-го раунда, и т.д. до тех пор, пока не узнает все вспомогательные ключи (или, что то же самое, весь мастер-ключ).

Рабочий пример. Проработаем «игрушечный» пример, показывающий, как можно найти хороший дифференциал. Используем четыре раунда SPN с длиной блока 16 битов на основе одного S-блока с длиной ввода/вывода 4 бита. S-блок определяется следующим образом (таблица показывает, как каждый 4-битный вход отображается на каждом 4-битном выходе):

Вход: 0I 00	0001	0010	0011	0100	0101	0110	0111
Выход:	0000	1011	0101	0001	0110	1000	1101
Вход: 1000	1001	1010	1011	1100	1101	1110	1111
Выход:	1111	0111	0010	1100	1001	0011	1110

Смешивающая перестановка, показывающая где каждый бит перемещается для каждого из 16 битов в блоке, выглядит следующим образом:

In:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Out:	7	2	3	8	12	5	11	9	10	1	14	13	4	6	16	15

x	S(x)	x ⊕ 1111	S(x ⊕ 1111)	S(x) ⊕ S(x ⊕ 1111)
0000	0000	1111	1010	**1010**
0001	1011	1110	1110	0101
0010	0101	1101	0011	0110
0011	0001	1100	1001	1000
0100	0110	1011	1100	**1010**
0101	1000	1010	0010	**1010**
0110	1101	1001	0111	**1010**
0111	0100	1000	1111	1011
1000	1111	0111	0100	1011
1001	0111	0110	1101	**1010**
1010	0010	0101	1000	**1010**
1011	1100	0100	0110	**1010**
1100	1001	0011	0001	1000
1101	0011	0010	0101	0110
1110	1110	0001	1011	0101
1111	1010	0000	0000	**1010**

РИС. 6.8: Влияние входной разности Δx = 1111 в нашем S-блоке.

Сначала найдем дифференциал в S-блоке. Пусть S(x) обозначает выход S-блока на входе x. Рассмотрим дифференциал Δx = 1111. Тогда, например, мы имеем S(0000) ⊕ S(1111) = 0000 ⊕ 1010 = 1010 и, таким образом, разность 1111 во входных данных приводит к разности 1010 в выходных данных. Посмотрим, часто ли это соотношение имеет место. Имеем S(0001) = 1011 и S(0001 ⊕ 1111) = S(1110) = 1110, и, таким образом, здесь разность 1111 во входных данных не приводит к разности в выходных данных. Тем не менее, S(0100) = 0110 и S(0100 ⊕1111) = S(1011) = 1100, и поэтому в данном случае разность 1111 во входных данных приводит к разности 1010 выходных данных. На рис. 6.8 мы имеем табличные результаты для всех возможных входных данных. Мы видим, что в половине случаев разность 1111 во входных данных дает разность 1010 в выходных данных. Таким образом, (1111, 1010) - это дифференциал в S , который появляется с вероятностью 1/2.

Выходная

Выходная разность	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	4	0	0	0	2	2	2	2	0	0	4	0
2	0	0	0	0	0	2	0	2	0	2	2	4	2	2	0	0
3	0	2	2	4	0	4	0	0	0	0	0	0	0	2	2	0
4	0	0	0	2	2	2	6	0	2	0	0	0	2	0	0	0
5	0	2	2	0	0	0	0	0	4	0	0	0	4	2	2	0
6	0	2	0	2	0	0	0	0	0	2	0	2	0	4	0	4
7	0	2	0	0	2	4	2	2	0	2	0	0	0	2	0	0
8	0	0	0	2	0	0	0	2	0	0	0	2	2	2	2	4
9	0	2	0	2	2	2	0	4	0	2	2	0	0	0	0	0
A	0	0	4	0	2	0	2	4	2	0	2	0	0	0	0	0
B	0	0	2	0	0	0	2	0	0	2	0	0	4	2	4	0
C	0	0	0	0	0	0	0	0	4	4	0	4	0	0	0	4
D	0	4	2	2	0	0	2	2	0	0	0	0	0	0	0	4
E	0	2	4	2	4	0	0	0	0	0	0	0	2	0	2	0
F	0	0	0	0	0	2	2	0	2	0	**8**	2	0	0	0	0

РИС. 6.9: Дифференциалы в нашем S-блоке.

Этот же процесс можно выполнить для всех 24 входных разностей Δx , чтобы вычислить вероятность каждого дифференциала. А именно, для каждой пары $(\Delta x, \Delta y)$ мы заносим в таблицу количество 4-битных входов x, для которых $S(x) \oplus S(x \oplus \Delta x) = \Delta y$. Мы сделали это для S-блока нашего примера на рис. 6.9. (Для краткости мы представим $(\Delta x, \Delta y)$, используя шестнадцатеричную запись). Таблицу нужно читать следующим образом: запись (i, j) подсчитывает, сколько входных данных с разностью i отображается в выходных данных с разностью j. Заметим, например, что имеется 8 входов с разностью 0xF = 1111, которая отображается на выходе 0xA = 1010, как мы показали выше. Это наиболее вероятный дифференциал (за исключением тривиального дифференциала (0, 0)). Но существуют другие интересные дифференциалы: входная разность 0x4 = 0100 отображается на выходной разности 0x6 = 0110 с вероятностью 6/16 = 3/8, и есть несколько дифференциалов с вероятностью 4/16 = 1/4.

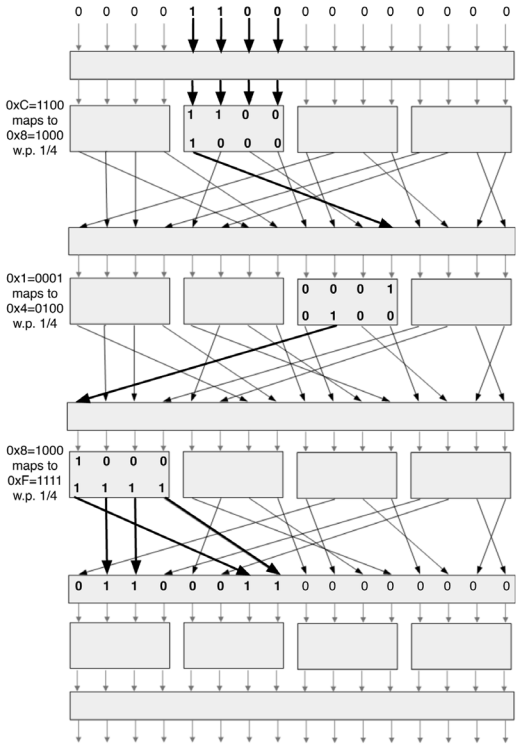

РИС. 6.10: Дифференциалы трассировки через четыре раунда SPN, в которой используется S-блок и смешивающая перестановка, приведенная в тексте.

Теперь продолжим это, чтобы найти хороший дифференциал для первых трех раундов SPN. Рассмотрим оценку SPN на двух входах, которые имеют дифференциал 0000 1100 0000 0000, и проследим дифференциал между промежуточными значениями на каждом этапе этой оценки. (См. рис. 6.10, на котором показаны четыре полных раунда SPN плюс этап заключительного ключевого смешивания. Для ясности, на рис. опущена смешивающая перестановка в 4-ом раунде; эта смешивающая перестановка влияет только на перетасовку битов дифференциала, и поэтому может быть легко учтена при атаке). Этап ключевого смешивания в первом раунде не влияет на дифференциал и поэтому входы для второго S-блока в первом раунде имеют дифференциал 1100. Из рис. 6.9. видно, что разность 0xC = 1100 во входных данных для S-блока дает разность 0x8 = 1000 в выходных данных S-блока с вероятностью 1/4. Таким образом, с вероятностью 1/4 дифференциал в выходных данных 2-го S-блока после раунда 1 является одиночным битом, который при смешивающей перестановке перемещается с 5-ой позиции в 12-ю позицию. (Входные данные для других S-блоков равны, поэтому их выходные данные равны и дифференциалы на выходах равны 0000). Если предположить, что это так, то входная разность для третьего S-блока во втором раунде равна 0x1 = 0001 (опять же, этап ключевого смешивания во втором раунде не влияет на дифференциал); используя рис. 6.9, имеем, что с вероятностью 1/4 выходная разность от этого S-блока равна 0x4 = 0100. Таким образом, опять же имеется только единственный выходной бит, который отличается, и он перемещается из 10-й позиции в первую позицию при смешивающей перестановке. И наконец, сверившись еще раз с рис. 6.9, видим, что входная разность 0x8 = 1000 для S-блока дает в результате выходную разность 0xF = 1111 с вероятностью 1/4. Биты в позициях 1, 2, 3 и 4 затем перемещаются при смешивающей перестановке в позиции 7, 2, 3 и 8.

В целом, мы далее видим, что входная разность Δx = 0000 1100 0000 0000 дает выходную разность Δy = 0110 0011 0000 0000 после трех раундов с вероятностью минимум 1/4•1/4 •1/4= 1/64•7 (Мы перемножаем вероятности, поскольку эвристически предполагаем независимость каждого раунда). Для случайной функции вероятность того, что появляется любой данный дифференциал равна только 2^{-16}=1/65536. Таким образом, дифференциал, который мы нашли, появляется с вероятностью существенно выше, чем можно было бы ожидать для случайной функции. Заметим также, что мы нашли дифференциал с низким весом.

Мы можем использовать этот дифференциал, чтобы найти первые 8 битов итогового вспомогательного ключа k5. Как обсуждалось ранее, начнем, пред

---

7 Это нижняя граница вероятности дифференциала, поскольку могут существовать другие разности в промежуточных значениях, которые приведут к той же разности в выходных данных.

ставляя, что $\{(x_i, x_i)\}_L$ является множеством L пар случайных входных данных с дифференциалом Δx. Используя атаку подобранного открытого текста, мы затем получим значения $y_i = F_k(x_i)$ и $y_i = F_k(x_i)$ для всех i. Теперь, для всех возможных значений для начальных 8 битов k5 мы вычислим начальные 8 битов $\tilde{y}_i$, $\tilde{y}_i$ промежуточные значения после этапа ключевого смешивания 4-го раунда. (Мы можем делать это потому, что нам нужно только инвертировать два крайних левых S-блока 4-го раунда, чтобы получить эти 8 битов). Если мы угадаем правильное значение для начальных 8 битов k5, то мы можем ожидать, что 8-битный дифференциал 0110 0011 появляется с вероятностью минимум 1/64. Эвристически неправильный прогноз дает ожидаемый дифференциал только с вероятностью $2^{-8} = 1/256$. При задании L достаточно большим, мы можем (с высокой вероятностью) определить правильное значение.

Дифференциальные атаки на практике. Дифференциальный криптоанализ – это очень мощное средство и оно было использовано для атаки реальных шифров. Известный пример представляет собой FEAL-8, который был предложен в качестве альтернативы DES в 1987 г. Дифференциальной атакой на FEAL-8 было установлено, что требуется только 1000 подобранных открытых текстов. В 1991 г. потребовалось менее 2 минут использования такой атаки, чтобы найти весь ключ. Сегодня любой предложенный шифр проверяется на устойчивость к дифференциальному криптоанализу.

Дифференциальная атака была первой атакой на DES, потребовавшей меньше времени, чем простой поиск последовательным перебором. Несмотря на интересный теоретический результат, эта атака не вызывает серьезной озабоченности на практике, поскольку для нее требуется 247 подобранных открытых текстов. Для взломщика очень сложно получить это множество текст/шифрованных пар подобранных в большинстве реальных приложений. Интересно отметить, что небольшие изменения S-блоков DES делают шифр гораздо более чувствительным к дифференциальным атакам. Личные свидетельства разработчиков DES (после дифференциальных атак, обнаруженных во внешнем мире) подтвердили, что S-блоки DES были специально разработаны, чтобы пресекать дифференциальные атаки.

Линейный криптоанализ. Линейный криптоанализ был разработан Мацуи (Matsui) в начале 1990-х гг. Мы только опишем метод на высоком уровне. Основная идея заключается в том, чтобы рассматривать отношения между входными и выходными данными, которые имеют более высокую вероятность, чем можно было бы ожидать от случайной функции.

Более подробно, говорят, что позиции битов $i_1, \ldots, i_{in}$ и $i^r, \ldots, i^r_{out}$ имеют линейный bias ε если, для однородных x and k, и $y \stackrel{def}{=} F(x)$, считается, что

$$\left|\Pr[x_{i_1} \oplus \cdots \oplus x_{i_{in}} \oplus y_{i'_1} \oplus \cdots \oplus y_{i'_{out}} = 0] - \frac{1}{2}\right| = \varepsilon,$$

где x_i, y_i обозначает i-ые биты x и y. Для любой случайной функции и любого фиксированного набора позиций битов мы ожидаем, что смещение будет близким к нулю. Мацуи показал, как использовать достаточно большое смещение в шифре F, чтобы найти секретный ключ. Помимо представленного другого метода для взлома шифров, важной особенностью этой атаки является то, что для нее не требуется подобранных открытых текстов, а достаточно известных открытых текстов. Это очень важно, поскольку зашифрованный файл может предоставить огромный объем известного открытого текста, тогда как сбор шифровок подобранных открытых текстов гораздо сложнее. Мацуи показал, что DES может быть взломан с помощью всего лишь 243 пар известный текст/шифрованный текст.

Воздействие на конструкцию блочного шифра. Современные блочные шифры, отчасти, разрабатываются и оцениваются на основе их устойчивости к дифференциальному и линейному криптоанализу. При конструировании блочного шифра разработчики выбирают S-блоки и другие компоненты так, чтобы минимизировать дифференциальные вероятности и линейные смещения. Заметим, что невозможно исключить все дифференциалы с высокой вероятностью в S-блоке: любой S-блок будет иметь какой-то дифференциал, который появляется более часто, чем другие. Тем не менее, эти отклонения можно свести к минимуму. Более того, увеличивая количество раундов (и тщательно выбирая смешивающую перестановку) можно как уменьшить дифференциальные вероятности, так и сделать шифр более трудным для криптоанализа, чтобы найти любые дифференциалы для использования.

6.3.Функции расстановки ключей (хэш-функции)

Напомним, из главы 5 известно, что первичное требование безопасности для хэш-функции H –это устойчивость к коллизиям; т.е. должно быть сложно найти коллизию или разные входы x, x^r , такие как, H(x) = H(x^r). (Опустим здесь упоминание любого ключа, поскольку реальные функции хэширования, как правило, отключены). Если функция хэширования имеет выход длиной A битов, то лучшее, на что можно надеяться, это то, что она должна быть недопустимой для обнаружения коллизии с использованием существенно меньшего, чем количества $2^{A/2}$ инициирований H. (См. раздел 5.4.1). Мы также хотели бы использовать функцию хэширования, чтобы добиться устойчивости (второго) прообраза против атак, протекающих за время намного меньше, чем 2^A , хотя в нашей дискуссии здесь мы не рассматриваем такие атаки.

Хэш-функции, как правило, создаются в два этапа. Сначала создают функцию сжатия (т.е. хэш-функцию фиксированной длины) h ; затем используется некий механизм, чтобы расширить h так, чтобы поддерживать входные данные про-

извольной длины. В разделе 5.2 мы уже показали один метод - преобразования Меркле-Дамгорда (Merkle–Damg˚ard) для второго этапа. Здесь мы рассматриваем методику для проектирования базисной функции сжатия. Мы также обсудим некоторые хэш-функции, используемые на практике. Теоретическое построение функции сжатия на основе теоретико-числового допущения приведено в разделе

6.3.1 Хэш-функции из блочных шифров

Может показаться удивительным, что можно создать устойчивую к коллизиям функцию сжатия из блочного шифра, который обладает определенными дополнительными свойствами. Есть несколько способов сделать это; один из наиболее распространенных с помощью конструкции Дэвиса-Мейера (Davies–Meyer). Пусть F будет блочным шифром с длиной ключа n-битов и длиной блока A- битов. Тогда можно определить функцию сжатия $h : \{0,1\}^{n+A} \rightarrow \{0,1\}^{A}$ by $h(k, x) \stackrel{\text{def}}{=} F(x) \oplus x$. (См. рис. 6.11.)

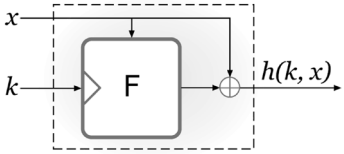

РИС. 6.11: Конструкция Мейера-Дэвиса.

Мы не знаем, как доказать устойчивость к коллизиям результирующей функции сжатия, основанной только на предположении, что F является строгой псевдослучайной перестановкой, и на самом деле, есть основания предполагать, что такое доказательство невозможно. Однако мы можем доказать устойчивость к коллизиям, если мы хотим смоделировать F в качестве идеального шифра. Модель идеального шифра представляет собой усиление модели случайного прогноза (см. раздел 5.5), в которой мы постулируем, что все стороны имеют доступ к прогнозу для случайной ключевой перестановки $F : \{0,1\}^n \times \{0,1\}^A \rightarrow \{0,1\}^A$ а также к ее инверсии F^{-1} (т.е., такой как $F^{-1}(k, F(k, x)) = x$ для всех k, x). Другой способ придумать это состоит в том, что каждый ключ $k \in \{0,1\}^n$ указывает независимую равномерную перестановку F (k, •) на A-битных строках. Как и в модели случайного прогноза есть только один способ вычислить F (или F^{-1}) – это явным образом запросить прогноз с (k, x) и получить обратно F (k, x) (или $F^{-1}(k, x)$).

Анализ конструкций в модели идеального шифра обладает всеми достоинствами и недостатками работы в модели случайного прогноза, как подробно обсуждается в разделе 5.5. Мы только добавим здесь, что модель идеального шифра предполагает отсутствие атак, связанных с ключом, в том смысле, что (как мы только что сказали) перестановки F (k, •) и F (k^r, •)должны вести себя независимо друг от друга даже, если, например, k и k^r отличаются только од-

ним битом. Кроме того, не должно быть никаких «плохих ключей» k (скажем, все нулевые кнопки), для которых F (k, •) легко отличить от случайных. Это также означает, что F (k, •) должны «вести себя случайно»даже когда k известно. Для любого реального шифра F эти свойства не обязательно поддерживаются (и даже четко не определены), даже если F является строгой псевдослучайной перестановкой и читатель может заметить, что мы не обсуждаем эти свойства в любом нашем анализе реальных конструкций блочных шифров. (На самом деле, DES и тройной DES не соответствуют этим характеристикам). Любой блочный шифр, используемый для создания образца идеального шифра, должен быть оценен относительно этих более жестких требований.

Докажем следующую теорему для конкретных условий, однако доказательство может быть легко адаптировано и для асимптотических условий.

ТЕОРЕМА 6.5 *Если F моделируется в качестве идеального шифра, то конструкция Мейера-Дэвиса дает устойчивую к коллизиям функцию сжатия. Точнее, любой взломщик, делая $q < 2^{A/2}$ запросов к прогнозам идеального шифра обнаруживает коллизию с вероятностью, чаще всего, $q^2/2^A$.*

ДОКАЗАТЕЛЬСТВО Чтобы было ясно, мы рассмотрим здесь вероятностный эксперимент, в котором F отбирается случайным образом (точнее, для каждого k $\in \{0,1\}^n$ функция $F(k,\bullet): \{0,1\}^A \to \{0,1\}^A$ выбирается равномерно из множества PermA перестановок A-битных строк) а затем взломщик получает доступ к прогнозу F и F^{-1}. Взломщик пытается затем найти пару с коллизией (k, x), (k^r, x^r), т.е. для которой $F(k,x) \oplus x = F(kr, x^r) \oplus x^r$. Никакие вычислительные ограничения не действуют на взломщика кроме ограничения количества запросов прогноза, которые он делает. Скажем, предполагаем, что если взломщик выводит пары с коллизией (k, x), (k^r, x^r), то он предварительно сделал запросы прогноза, необходимые для вычисления значений F (k, x) и F (k^r, x^r). Мы также предполагаем, что взломщик никогда не делает один и тот же запрос более, чем один раз и никогда не запрашивает $F^{-1}(k, y)$, если уже узнал, что y = F (k, x) (и наоборот). Все эти предположения сделаны без ограничения общности.

Рассмотрим i-й запрос взломщика, сделанный по его прогнозам. Запрос (ki, xi)к F показывает только значение хэш-функции $hi \stackrel{def}{=} h(k_i, x_i) = F(k_i, x_i) \oplus xi$; аналогично, запрос к F^{-1}, дающий результат $xi = F_{-1}(k_i, y_i)$, дает только значение хэш-функции $hi \stackrel{def}{=} h(ki, xi) = yi \oplus F^{-1}(ki, yi)$. Взломщик не получит коллизии кроме $h_i = h_j$ для некоторых i *f*= j.

Зафиксируем i, j при i > j и рассмотрим вероятность того, что hi = hj . Во время i-го запроса значение hj зафиксировано. Коллизия между hi и hj получается на i-м запросе, только если взломщик обращается с запросом (ki, xi) к F и получает результат F (ki, xi) = hj ⊕ xi, или обращается с запросом (ki, yi) к F^{-1} и получает

результат $F^{-1}(ki, yi) = hj \oplus yi$. Либо событие происходит с вероятностью, чаще всего, $1/(2^A - (i-1))$ поскольку, например, $F(k_i, x_i)$ равномерна по $\{0,1\}^A$, за исключением того, что она не может быть равна любому значению $F(ki, x)$ уже определенному взломщиком (чаще всего) $i-1$ предыдущими запросами прогноза с использованием ключа k_i. Поскольку $i \leq q < 2^{A/2}$, вероятность того, что $hi = hj$ чаще всего равна $2/2^A$.

Принимая границу объединения по всем .q/2. $< q^{2/2}$ различным парам i, j дает результат, указанный в теореме.

Davies–Meyer and DES. Как упоминалось выше, при реализации конструкции Дэвиса-Мейера следует соблюдать осторожность с любым реальным блочным шифром, поскольку шифр должен обладать дополнительными свойствами (помимо пригодности к строгой псевдослучайной перестановке), чтобы в результате построения быть безопасным. В упражнении 6.21 мы исследуем, что происходит не так, когда DES используется в конструкции Дэвиса-Мейера.

Это должно служить предупреждением, что доказательство безопасности для конструкции Дэвиса-Мейера в модели идеального шифра не обязательно преобразуется в реальную безопасность, если реализуется пример с реальным шифром. Тем не менее, как мы опишем ниже, эта парадигма была использована для построения практических хэш-функций, которые противостояли атаке (а конкретно, когда блочный шифр в центре конструкции был предназначен специально для этой цели).

В конструкции Дэвиса-Мейера есть полезная парадигма для конструирования функций сжатия, устойчивых к коллизиям. Однако ее не следует применять к блочным шифрам, предназначенным для шифрования, подобно DES и AES.

6.3.2 MD5

MD5 - это хэш-функция с длиной выходных данных 128 битов. Она была разработана в 1991 г. и в течение некоторого времени считалась устойчивой к коллизиям. В течение нескольких лет в MD5 начали обнаруживаться различные недостатки, но они, по всей видимости, не приведут к какому-либо простому способу поиска коллизий. Поразительно, но в 2004 году группа китайских криптоаналитиков представила новый метод нахождения коллизий в MD5; они легко смогли убедить других, что их подход был правильным, демонстрируя недвусмысленную коллизию! С тех пор атака была улучшена и сейчас коллизии можно обнаружить в течение одной минуты на настольном ПК. Кроме того, атаки были расширены таким образом, что можно было обнаружить даже «контролируемые коллизии» (например, два файла в формате postscript, генерирующих видимый контент). Из-за этих атак MD5 не следует использовать нигде, где требуется криптографическая защита. Отметим MD5 только потому, что она по-прежнему обнаруживается в унаследованном программном коде.

6.3.3 SHA-0, SHA-1 и SHA-2

Безопасный хэш-алгоритм (Secure Hash Algorithm (SHA)) относится к серии криптографических хэш-функций, стандартизованных NIST. Пожалуй, наиболее известен из них SHA-1, который был введен в 1995 г. Этот алгоритм имеет выход длиной 160 битов и вытеснил предшественника, называемого SHA-0, от которого отказались из-за неуказанных недостатков, обнаруженных в этом алгоритме.

На момент написания этой книги явную коллизию еще предстоит найти в SHA-1. Однако теоретический анализ в течение последних нескольких лет указывает на то, что коллизии можно обнаружить, используя существенно меньше, чем 280 оценок хэш-функции, которые потребовались бы при использовании криптоанализа на основе парадокса дней рождения, и он позволил предположить, что коллизия будет найдена в ближайшее время. Поэтому рекомендуется перейти на SHA-2, который, по всей видимости, в настоящий момент не имеет тех же недостатков. SHA-2 состоит из двух связанных функций: SHA-256 и SHA-512 с длиной выходных данных 256 и 512 битов соответственно.

Все хэш-функции в семействе SHA построены с использованием той же базовой конструкции, которая содержит компоненты, которые мы уже видели: Сначала определяют функцию сжатия, применяя конструкцию Дэвиса-Мейера к блочному фильтру, а затем ее расширяют для поддержки произвольной длины входных данных, используя преобразования Меркле-Дамгорда (Merkle–Damg˚ard). Одним интересным фактом здесь является то, что блочный шифр в каждом случае был разработан специально для создания функции сжатия. На самом деле лишь задним числом основные компоненты в функциях сжатия были выделены и проанализированы в качестве блочных шифров SHACAL-1 (для SHA-1) и SHACAL-2 (для SHA-2). Эти шифры интригующие сами по себе, поскольку имеют большую длину блока (160 и 256 битов соответственно) и 512-битные ключи.

6.3.4 SHA-3 (Keccak)

В период после атаки коллизии на MD5 и теоретических недостатков, обнаруженных в SHA-1, NIST в конце 2007 г. объявил открытый конкурс на разработку новой криптографической хэш-функции, которая будет названа SHA-3. Представленные алгоритмы были необходимы для поддержки, как минимум, длины выходных данных 256 и 512 битов. Как и в случае AES приблизительно 10 лет назад, конкурс был полностью открытым и прозрачным; любой мог представить алгоритм для рассмотрения, и общественности было предложено выразить свое мнение по любому из вариантов. В первом туре количество кандидатов сократилось с 51 до 14 в декабре 2008 г. и далее их круг сузился до пяти финалистов в 2010 г. Оставшиеся кандидаты были объектом пристального изучения со стороны криптографического сообщества в течение следующих двух лет. В октябре

2012 г. NIST объявил о выборе Кеччака (Keccak) в качестве победителя конкурса. По состоянию на момент написания этой работы данный алгоритм подвергается стандартизации в качестве следующего поколения для замены SHA-2.

Кеччак необычен в нескольких отношениях. (Интересно отметить, что одной из причин, по которой Кеччак был выбран, является то, что его структура сильно отличается от структуры SHA-1 и SHA-2). По своей сути она основана на бесключевой перестановке f с большой длиной блока 1600 битов; это радикально отличается, например, от конструкции Дэвиса-Мейера, которая основана на ключевой перестановке. Кроме того, Кеччак не использует преобразование Меркле-Дамгорда, чтобы поддерживать произвольную длину входных данных. Вместо этого он использует новый подход, называемый губчатой конструкцией. Кеччак и губчатая конструкция могут быть проанализированы в более общем плане в модели случайной перестановки, в которой мы постулируем, что стороны имеют доступ к прогнозу для случайной перестановки $f: \{0,1\}^A \rightarrow \{0,1\}^A$ (и, возможно, к ее инверсии). Это менее строго, чем для модели идеального шифра; на самом деле мы можем легко получить случайную перестановку в модели идеального шифра посредством простой фиксации ключа для шифра, чтобы было любое постоянное значение.

Будет интересно наблюдать, как развивается новый хэш-стандарт и видеть, насколько быстро разработчики адаптируются от SHA-1/SHA-2 к новому SHA-3.

Ссылки и дополнительная литература

Дополнительную информацию по регистрам сдвига с линейной обратной связью можно найти в

Handbook of Applied Cryptography [120] (Справочник по прикладной криптографии) или в более новом тексте Паара (Paar) и Пельцля (Pelzl) [135]. Дальнейшие подробности, относящиеся к eSTREAM, а также подробную спецификацию Trivium, можно найти по адресу http://www.ecrypt.eu.org/stream.

Обзор атак на RC4 см. в работе Альфардан и др. (AlFardan et al.) [9].

Парадигма смешения-диффузии и сети замены-перестановки были введены Шэнноном (Shannon) [154] и Фейстелем (Feistel) [64]. Более подробную информацию, касающуюся дизайна SPN, см. в тезисах Хейса (Heys) [90]. Общие атаки на трехраундовые SPN, которые лучше чем те, которые мы показали здесь, известны из работы [31]. Майлс (Miles) и Виола (Viola) [126] провели теоретический анализ сетей SPN.

Сети Фейстеля (Feistel) были впервые описаны в [64]. Теоретический анализ сетей Фейстеля выполнили Люби (Luby) и Ракофф (Rackoff); см. гл. 7.

Более подробную информацию по DES, AES и конструкциях блочных шиф-

ров в целом , можно найти в тексте Кнудсена (Knudsen) и Робшоу (Robshaw) [106]. Атака встречи посередине на двойное шифрование выполнена Диффи (Diffie) и Хеллманом (Hellman) [59]. Атака на тройное шифрование с двумя ключами упомянутыми в тексте (и исследованными в упражнении 6.13) выполнена Меркле (Merkle) и Хеллманом (Hellman) [124]. Теоретический анализ безопасности двойного и тройного шифрования можно найти в [6, 24].

DESX представляет собой другую технику для увеличения эффективной длины ключа DES. Секретный ключ состоит из значений $k_i, k_o \in \{0,1\}^{64}$, и $k \in \{0,1\}^{56}$, и шифр определяется

$DESX_{ki,k,ko}(x) \stackrel{def}{=} ko \oplus DESk(x \oplus k_i)$.

Эту методологию впервые изучили Эвен (Even) и Мансур (Mansour) [63] в несколько отличном контексте. Ее применение к DES было предложено в неопубликованной работе Ривестом (Rivest), а ее безопасность позднее анализировали Килиан (Kilian) и Рогэвэй (Rogaway) [105, 149]. Дифференциальный криптоанализ ввели Бихэм (Biham) и Шамир (Shamir) и его применение к DES описано в книге этих авторов [30]. Копперсмит (Coppersmith) [45] описывает принципы проектирования S-блоков DES в свете представления общественности дифференциального криптоанализа. Линейный криптоанализ был представлен Мацуи в работе [118], в которой показано его применение к DES. Для получения более подробной информации об этих передовых криптоаналитических методах мы отсылаем читателя к учебнику по дифференциальному и линейному криптоанализу [91] или к упомянутой выше книге Кнудсена (Knudsen) и Робшоу (Robshaw) [106].

Дополнительную информацию о MD5 и SHA-1 см. в [120]. Однако следует заметить, что их изложению предшествуют атаки Вонга и др. (Wang et al.) [175, 174]. Конструкции функций сжатия из блочных шифров анализируются в [143, 33]. Губчатая конструкция описана и проанализирована Бертони и др. (Bertoni et al.) [28]. Дополнительную информацию о конкурсе SHA-3 см. на веб-сайте NIST по адресу:

http://csrc.nist.gov/groups/ST/hash/sha-3/index.html.

Упражнения

Предположим, что имеется 6-ступенчатый LFSR (регистр сдвига с линейной обратной связью) с $c_0 = c_5 = 1$ and $c_1 = c_2 = c_3 = c_4 = 0$.

(a) Каковы первые 10 битов выходных данных у этого LFSR если в исходном состоянии они начинаются с (1, 1, 1, 1, 1, 1)?

(b) Это максимальная длина LFSR?

В этой задаче мы рассмотрим нелинейный комбинационный генератор, где

имеется n -ступенчатый LFSR, но выход на каждом временном шаге не s0, а вместо этого $g(s_0, \ldots, s_{n-1})$ для некоторой нелинейной функции g. Предположим, что коэффициенты обратной связи LFSR известны, а его начальное состояние - нет. Покажите, что каждый из следующих вариантов g не позволяет получить хороший псевдослучайный генератор:

(a) $g(s_0, \ldots, s_{n-1}) = s_0 \wedge s_1$.

(b) $g(s_0, \ldots, s_{n-1}) = (s_0 \wedge s_1) \oplus s_2$.

Пусть F будет блочным шифром с длиной ключа и длиной блока n битов. Допустим есть атака с регенерацией ключа на F, которая успешна с вероятностью 1 с использованием n подобранных открытых текстов и минимальными вычислительными затратами. Докажите, что формально F не может быть псевдослучайной перестановкой.

В нашей атаке на одноraундовую SPN мы рассматривали блок длиной 64 бита и 16 S-блоков, каждый из которых имеет 4-битный вход. Повторите анализ для случая 8 S-блоков, каждый из которых имеет 8-битный вход. Какова теперь сложность атаки? Снова повторите анализ с длиной блока 128 битов и 16 S-блоками, каждый из которых имеет 8-битный вход.

Рассмотрите модифицированную SPN, где вместо выполнения этапов ключевого смешивания, замены и перестановки в чередующемся порядке для r (полных) раундов, шифр вместо этого применяет r рауднов ключевого смешивания, затем выполняет r раундов замены и, наконец, применяет r смешивающих перестановок. Проанализируйте безопасность этой конструкции.

В этой задаче мы предполагаем двухраундовую SPN с длиной блока 64 бита.

(a) Предположим, что в каждом раунде используются вспомогательные 64-битные ключи, так что мастер-ключ имеет длину 192 бита. Покажите атаку с регенерацией ключа, использующую менее 2192 попыток.

(b) Предположим, что первый и третий вспомогательные ключи одинаковы, а второй вспомогательный ключ независимый, так что мастер-ключ имеет длину 128 битов. Покажите атаку с регенерацией ключа, использующую намного менее 2^{128} попыток.

Каков выход r-раундовой сети Фейстеля, если на входе (L_0, R_0) в каждом из следующих двух случаев:

(a) Каждая раундовая функция выводит все нули, независимо от входных данных.

(b) Каждая раундовая функция является тождественной функцией.

Пусть Feistelf1 ,f2 (•) означает двухраундовую сеть Фейстеля, использующую функции f_1 и f_2 (в этом порядке). Покажите, что если Feistelf1 ,f2 $(L_0, R_0) = (L_2, R_2)$, то Feistelf2 ,f1 $(R_2, L_2) = (R_0, L_0)$.

Для этого упражнения руководствуйтесь описанием DES, приведенным в

этой главе, но используйте тот факт, что в реальной конструкции DES обе эти половины выхода конечного раунда сети Фейстеля, поменялись местами. Т.е. если выход конечного раунда сети Фейстеля (L16, R16), то выход DES - (R16, L16).

(a) Покажите, что единственная разница между вычислением DESk и DES−1- это порядок, в котором используются вспомогательные ключи. (Опирайтесь на предыдущее упраждение).

(b) Покажите, что для $k = 0^{56}$ считается, что $DES_k (DES_k (x)) = x$.

Подсказка: Считайте, что вспомогательные ключи генерируются из этого ключа.

(c) Найдите три других ключа DES с тем же свойством. Эти ключи известны как плохие ключи для DES. (Примечание: ключи, которые вы найдете, будут отличаться от реальных плохих ключей DES из-за различий в вашем представлении).

(d) Представляют ли эти 4 плохих ключа серьезную уязвимость в использовании тройного DES, в качестве псевдослучайной перестановки? Объясните.

Покажите, что DES имеет такое свойство, что $DESk (x) = \overline{DESk}^{-} (x)$ для каждого ключа k и вход x (где z^{-} означает побитовое дополнение z).

(Это называется свойством комплементарности DES). Может ли это представлять серьезную уязвимость в использовании тройного DES в качестве псевдослучайной перестановки? Объясните.

Опишите атаки на следующие модификации DES:

(a) Каждый вспомогательный ключ имеет длину 32 бита, а раундовая функция является просто исключающей ИЛИ вспомогательного ключа входа раунда (т.е. $f(k, R) = k_i \oplus R$). Для этой задачи, список ключей не важен и вспомогательные ключи ki можно рассматривать как независимые ключи.

(b) Вместо использования разных вспомогательных ключей в каждом раунде используется один и тот же 48-битный вспомогательный ключ. Покажите, как отличить шифр от случайной перестановки 248 раз.

Подсказка: Могут помочь упражнения 6.8 и 6.9. . .

(Это упражнение основано на упражнении 6.9). Наша цель в том, чтобы показать, что для любого плохого ключа k DES легко найти вход x такой, что DESk (x) = x.

(a) Предположим, что мы оцениваем DES_k на входе (L_0, R_0), и выход после 8 раундов сети Фейстеля (L_8, R_8) при $L_8 = R_8$. Покажите, что выход $DES_k (L_0, R_0)$ - это (L_0, R_0). (Напомним из упражнения 6.9, что DES меняет местами две половины 16-го раунда сети Фейстеля перед выводом результата).

(b) Покажите, как найти вход (L_0, R_0) со свойством в части (a).

Эта задача иллюстрирует атаку на тройное шифрование с двумя ключами.

Пусть F будет блочным шифром с n-битной длиной блока и ключа, и задает $F'_{k_1,k_2}(x) \stackrel{\text{def}}{=} F_{k_1}(F_{k_2}^{-1}(F_{k_1}(x)))$.

(a) Предположим, что с данной парой (m_1, m_2) можно найти, за постоянное время, все ключи k_2, так что $m_2 = F_{-1}(m_1)$. Покажите, как восстановить весь ключ для F^r (с высокой вероятностью) приблизительно за время 2n, используя три известных пары вход/выход.

(b) В общем, будет невозможно найти k_2, как указано выше, в течение постоянного времени. Тем не менее, покажите, что при использовании этапа предварительной обработки, требующей времени 2^n, можно, задав m_2, найти (в принципе) в течение постоянного времени все ключи k_2 так, чтобы $m_2 = F^{-1}(0^n)$.

(c) Предположим, что k_1 известно, и что указанный выше этап предварительной обработки уже запущен. Покажите, как использовать значение $y = F^r(x)$ для единственного выбранного открытого текста x, чтобы определить k_2 в течение постоянного времени.

(d) Поместите указанные выше компоненты вместе, чтобы разработать атаку, которая восстанавливает весь ключ F^r при выполнении, приблизительно в течение времени 2^n и запрашивании шифрования примерно 2n выбранных входов.

Допустим, что список ключей DES модифицируется следующим образом: левая половина мастер-ключа используется, чтобы получить все вспомогательные ключи в раундах 1-8, тогда как правая половина мастер-ключа используется, чтобы получить все вспомогательные ключи в раундах 9–16. Покажите атаку на эту модифицированную схему, которая восстанавливает весь ключ приблизительно за время 2^{28}.

6.15 Пусть $f:\{0,1\}^m \times \{0,1\}^A \rightarrow \{0,1\}^A$ и $g:\{0,1\}^n \times \{0,1\}^A \rightarrow \{0,1\}^A$ будут безопасными блочными шифрами с m > n, и определим $Fk_1,k_2(x) = fk_1(gk_2(x))$. Покажите атаку с регенерацией ключа F с использованием времени $O(2^m)$ и пространства $O(A \cdot 2^n)$.

Определите $DESY_{k,kt}(x) = DES_k(x \oplus k_r)$. Длина ключа DESY равна 120 битам. Покажите атаку на DESY с регенерацией ключа, требующую времени и пространства $\approx 2^{64}$.

Выберите случайные S-блоки и смешивающие перестановки для сетей SPN разных размеров и разработайте дифференциальные атаки на них. Мы рекомендуем попробовать пятираундовые SPN с блоками длиной 16 и 24 бита, используя S-блоки с 4-битным входом/выходом. Запишите код, чтобы вычислить таблицы дифференциалов и провести атаку.

Обеспечьте компромисс по времени/пространству для 40-битного DES (т.е. зафиксируйте первые 16 битов ключа DES на нуле). Рассчитайте необходимые

время и память, и эмпирически оцените вероятность успеха. Экспериментально проверьте увеличение вероятности успеха, поскольку количество таблиц увеличивается. (Предупреждение: это большой проект).

Для каждой из следующих конструкций функции сжатия h из блочного шифра F, либо покажите атаку, либо докажите устойчивость к коллизиям в модели идеального шифра:

(a) $h(k, x) = F_k(x)$.

(b) $h(k, x) = F_k(x) \oplus k \oplus x$.

(c) $h(k, x) = F_k(x) \oplus k$.

Рассмотрите использование DES, чтобы сконструировать функцию сжатия следующим образом: Определите $h : \{0,1\}^{112} \rightarrow \{0, 1\}^{64}$ как $h(x,x) \stackrel{def}{=}$ DES (DES (064)) где $|x1| = |x2| = 56$.

(a) Запишите явную коллизию в h.

Подсказка: Используйте упражнение $6.9_{(a–b)}$.

(b) Покажите, как найти прообраз произвольного значения y (т.е., x_1, x_2 так, что $h(x1||x2) = y$) примерно в течение времени 2^{56}.

(c) Покажите более умную атаку прообраза, которая проходит приблизительно в течение времени 2^{32} и успешна с высокой вероятностью.

Подсказка: Опирайтесь на результаты Приложения A.4.

Пусть F будет блочным шифром, для которого легко найти фиксированные точки для некоторого ключа: а именно, есть ключ k , для которого легко найти входы x , для которых $F_k(x) = x$. Найдите коллизию в конструкции Дэвиса-Мейера при применении к F . (Рассмотрите это в свете упражнения 6.12).

Глава 7

*Теоретические конструкции примитивов симметричного ключа

В главе 3 мы ввели понятие псевдослучайности и определили некоторые основные криптографические примитивы, включая псевдослучайные генераторы, функции и перестановки. В главах 3 и 4 мы показали, что эти примитивы служат в качестве строительных блоков для криптографии с закрытым ключом. Как таковая, она имеет важное значение для понимания этих примитивов, с теоретической точки зрения. В этой главе мы формально введем понятие односторонних функций — функций, которые, неформально, легко вычислить, но трудно инвертировать, и покажем, как псевдослучайные генераторы, функции и перестановки могут быть сконструированы при единственном допущении, что односторонние функции существуют. [1] Кроме того, мы увидим, что односторонние функции необходимы для «нетривиальной» криптографии с закрытым ключом. То есть: существование односторонних функций эквивалентно существованию всей (нетривиальной) криптографии с закрытым ключом. Это одно из главных достижений современной криптографии.

Конструкции, которые мы покажем в этой главе, следует рассматривать как дополнения к конструкциям потоковых шифров и блочных шифров, обсуждавшихся в предыдущей главе. В центре внимания предыдущей главы было то, как различные криптографические примитивы реализуются в настоящее время на практике, целью этой главы было ввести некоторые используемые практические подходы и принципы проектирования. Несколько разочаровывает тот факт, что ни одна из показанных нами конструкций не доказала безопасность, основанную на слабых (т.е., более разумных) предположениях. В отличие от этого, в настоящей главе мы докажем, что возможно построить псевдослучайные перестановки, начиная с очень мягкого допущения, что односторонняя функция существует. Это предположение более приемлемо, чем, скажем, допущение, что AES является псевдослучайной перестановкой, как потому, что это качественно более слабое допущение, так и потому, что у нас есть ряд вариантов теоретико-числовых функций, которые изучались в течение многих лет, даже до появления криптографии. (Дальнейшее обсуждение этого пункта см. в самом начале главы 6). Однако недостатком является то, что все конструкции, показанные нами здесь, гораздо менее эффективны, чем конструкции главы 6 и, поэтому, фактически не используются. Остается важный вызов для криптографов, чтобы «преодолеть разрыв» и разработать доказуемые безопасные

[1] Это не совсем верно, поскольку, мы, по большей части, собираемся опираться на односторонние перестановки в этой главе. Но известно, что односторонних функций достаточно.

конструкции псевдослучайных генераторов, функций и перестановок, сравнимых по эффективности с лучшими доступными поточными шифрами и блочными шифрами.

Устойчивые к коллизиям хэш-функции. В отличие от предыдущей главы, мы здесь не рассматриваем устойчивые к коллизиям хэш-функции. Причина заключается в том, что конструкции таких хэш-функций из односторонних функций неизвестны и, на самом деле, имеются свидетельства, что такие конструкции невозможны. Мы обратимся к доказуемым конструкциям устойчивых к коллизиям хэш-функций, основанных на специфических теоретико-числовых допущениях , в разделе 8.4.2.

Замечание по поводу этой главы. Материал в этой главе более современный, чем материал в остальной части этой книги. Данный материал не используется в явном виде где-нибудь еще и, при желании, эту главу можно пропустить. Сказав это, мы попытались представить материал таким образом, чтобы он был понятным (с трудом) способным студентам последнего курса или начинающим аспирантам. Мы призываем всех читателей внимательно изучить разделы 7.1 и 7.2, в которых вводятся односторонние функции, и представлен обзор остальной части этой главы. Мы считаем, что знакомство, по крайней мере, с некоторыми из тем, рассматриваемых здесь, достаточно важно, чтобы оправдать затраченные усилия.

7.1 Односторонние функции

В этом разделе мы формально определим односторонние функции, а затем кратко обсудим некоторые варианты, которые согласно широко распространенному мнению, соответствуют этому определению. (Больше примеров предполагаемых односторонних функций мы увидим в главе 8). Далее мы введем понятие трудных предикатов, которое можно рассматривать как выделение в отдельный элемент сложности инвертирования односторонней функции, и которое будет широко использоваться в конструкциях, рассматриваемых в следующих разделах.

Определения

Одностороннюю функцию $f: \{0, 1\}^* \rightarrow \{0, 1\}^*$ легко вычислить, но сложно инвертировать. Первое условие легко формализовать: мы просто потребуем, чтобы f можно было вычислить за полиномиальное время. Поскольку мы заинтересованы в создании криптографических схем, которые трудны для взлома вероятностного полиномиально-временного противника, за исключением пренебрежимо малой вероятности, мы формализуем второе условие, потребовав, чтобы для любого вероятностного полиномиально-временного алгоритма было невозможно инвертировать f, т.е. найти прообраз данного значения y, за исключением пренебрежимо малой вероятности. Технический вопрос заключается в том, что эта вероятность берется из эксперимента, в котором y генерируется за счет выбора однородного элемента x области f и затем задания $y := f(x)$ (а не выбора y равномерно из диапазона f). Причина этого должна стать понятной из

конструкций, которые мы увидим в остальной части главы.

Пусть f : будет функцией {0, 1}* → {0, 1}*. Рассмотрим следующий эксперимент, определенный для любого алгоритма A и любого значения f для параметра безопасности.

Инвертирующий эксперимент InvertA,f (n)

1. Выберите однородные x ∈ {0, 1}^{n}, и вычислите y := f (x).

2. A задано равным 1^n , а y как вход и выходы x^r .

3. Выход эксперимента определяется как 1, если f (x^r) = y, и 0 в противном случае.

Подчеркнем, что для A не нужно искать оригинальный прообраз x; для A достаточно найти любое значение xr для которого f (xr) = y = f (x). Мы задаем параметр безопасности 1n для A на втором этапе, чтобы подчеркнуть, что A может действовать в течение полиномиального времени в параметре безопасности n независимо от длины y.

Теперь определим, что это значит для функции f, чтобы она была односторонней.

ОПРЕДЕЛЕНИЕ 7.1 *Функция f:{0,1}*→{0,1}* является односторонней, если выполняются следующие условия:*

1. **(Несложно вычислить:)** *Имеется полиномиально-временной алгоритм Mf, вычисляющий f ; то есть, Mf (x) = f (x) для всех x.*

2. **(Сложно инвертировать)** *Для любого вероятностного полиномиально-временного алгоритма*

A существует пренебрежимо малая функция negl, такая что

Pr[Invert$_{A,f}$ (n) = 1] ≤ negl(n).

Примечание. В этой главе мы часто будем делать вероятностное пространство более определенным, индексируя (часть) его в вероятностной записи. Например, мы можем кратко выразить второе требование в определении, указанном выше, следующим образом: Для каждого вероятностного полиномиально-временного алгоритма A существует пренебрежимо малая функция negl, такая что

$$\Pr_{x \leftarrow \{0,1\}^n}\left[\mathcal{A}(1^n, f(x)) \in f^{-1}(f(x))\right] \leq \mathsf{negl}(n)$$

(Напомним, что x ← {0, 1}^{n} означает, что x выбирают равномерно из {0, 1}^{n}). Вероятность, указанная выше, также берется по случайному распределению, используемому A, что остается здесь неясным.

Успешная инверсия односторонних функций. Функцию, которая не является односторонней, не обязательно легко инвертировать всегда (или даже

«часто»). Скорее наоборот, второе условие Определения 7.1 заключается в том, что существует вероятностный полиномиально-временной алгоритм A и не пренебрежимо малая функция γ, такая что A инвертирует $f(x)$ с вероятностью, минимум, $\gamma(n)$ (где вероятность выбирают по равномерному из $x \in \{0, 1\}^n$ и случайному A). Это, в свою очередь, означает, что существует положительный многочлен $p(\bullet)$ такой, что для бесконечного множество значений n алгоритм A инвертирует f с вероятностью не менее $1/p(n)$. Таким образом, если существует алгоритм A, который инвертирует f с вероятностью n^{-10} для всех четных значений n (но никогда не может инвертировать f, если n нечетное), то f не является односторонней функцией—даже в том случае, когда A успешно срабатывает на половине значений n и срабатывает только с вероятностью n^{-10} (для значений n, где он вообще срабатывает).

Эспоненциально-временная инверсия. Любую одностороннюю функцию можно инвертировать в любой точке y в экспоненциальном времени, просто проверяя все значения $x \in \{0, 1\}^n$ до тех пор, пока не будет найдено такое значение x, что $f(x) = y$. Таким образом, существование односторонних функций - это, по своей сути, предположение о вычислительной сложности и вычислительной трудности. То есть, речь идет о проблеме, которую можно решить в принципе, но предполагается, что ее сложно решить эффективно.

Односторонние перестановки. Нас часто будут интересовать односторонние функции с дополнительными структурными свойствами. Мы говорим, что функция f является сохраняющей длину, если $|f(x)| = |x|$ для всех x. Односторонняя функция, которая сохраняет длину и сохраняется один-к-одному, называется односторонней перестановкой. Если f является односторонней функцией, то любое значение y имеет уникальный прообраз $x = f^{-1}(y)$. Тем не менее, трудно найти x в течение полиномиального времени.

Семейства односторонних функций/перестановок. Приведенные выше определения односторонних функций и перестановок удобны тем, что в них рассматривается единственная функция в бесконечной области и диапазоне. Однако большинство вариантов односторонних функций и перестановок не вписываются в эти рамки. Вместо этого существует алгоритм, генерирующий некоторое множество параметров I, который определяет функцию f_I ; односторонность означает здесь, по существу, что f_I должна быть односторонней, с практически пренебрежимо малой вероятностью, при всех I. Поскольку каждое значение I определяет разные функции, мы теперь говорим о семействах односторонних функций (соотв., перестановок). Теперь приведем определение и отошлем читателя за конкретным примером к следующему разделу (см. также раздел 8.4.1).

ОПРЕДЕЛЕНИЕ 7.2 *Запись $\Pi = (Gen, Samp, f)$ вероятностного полиномиально временного алгоритма является семейством функций, если выполняются следующие условия:*

1. Алгоритм генерации параметров Gen, *на входе 1^n, выходные параметры I при $|I| \geq n$. Каждое значение I выхода Gen определяет множества DI и RI , которые устанавливают область и диапазон, соответственно, функции fI .*

2. Алгоритм выборки Samp, *на входе I, выводит равномерно распределенный элемент DI .*

3. Детерминированный алгоритм оценки f , *на входе I и $x \in DI$, выводит элемент $y \in RI$. Мы записываем это как $y := fI(x)$.*

П – это семейство перестановок, если для каждого значения I выхода Gen(1^n), считается, что $D_I = R_I$ и функция $fI : D_I \rightarrow D_I$ является взаимно однозначным соответствием.

Пусть П будет семейством функций. То, что отсюда следует – это естественная аналогия эксперимента, введенного ранее.

Инвертирующий эксперимент $\mathbf{Invert_{A,П}(n)}$:

1. *Gen(1^n) выполняется, чтобы получить I, а затем выполняется Samp(I), чтобы получить однородные $x \in DI$. В конце вычисляется $y := fI(x)$.*

2. А при заданных I и y на входе выводит xr .

3. Выходные данные эксперимента - это 1, если $fI(xr) = y$.

ОПРЕДЕЛЕНИЕ 7.3 *Семейство функций/перестановок П = (Gen, Samp, f) является односторонним, если для всех вероятностных полиномиально-временных алгоритмов А существует пренебрежимо малая функция negl, такая что*

$\Pr[\mathrm{Invert}_{A,П}(n) = 1] \leq \mathrm{negl}(n)$.

В этой главе мы работаем с односторонними функциями/перестановками в бесконечной области (как в Определении 7.1), но не работаем с семействами односторонних функций/перестановок. Это делается, в первую очередь, для удобства и существенно не влияет ни на какие результаты (см. упражнение 7.7).

Варианты односторонних функций

Односторонние функции интересны, только если они существуют. Мы не знаем, как доказать, что они обязательно существуют (это было бы большим прорывом в теории сложности), поэтому мы должны предположить или допустить их существование. Такое предположение основано на том, что некоторым естественным вычислительным проблемам было уделено больше внимания, но по-прежнему не имеется полиномиально-временного алгоритма для их решения. Возможно, наиболее известной из таких проблем является целочисленная факторизация, т.е. поиск простых множителей для большого целого числа. Очень просто перемножить два числа и найти их произведение, но сложно взять число и определить его множители. Это подводит нас к тому, чтобы определить функцию $\mathrm{fmult}(x, y) = x \cdot y$. Если мы не накладываем никаких ограничений на длину x и y, то fmult несложно инверти-

ровать: с высокой вероятностью x • y будет четным, в этом случае $(2, x^{y/2})$ является инверсией. Эта проблема может быть решена ограничением области fmult до простых чисел равной длины x и y. Мы вернемся к этой идее в разделе 8.2.

Другой вариант односторонней функции, не опирающийся непосредственно на теорию чисел, основан на проблеме суммы подмножеств и определяется

$$f_{\mathrm{ss}}(x_1, \ldots, x_n, J) = \left(x_1, \ldots, x_n, \left[\sum_{j \in J} x_j \bmod 2^n\right]\right)$$

где каждое x_i является n-битной строкой, интерпретируемой в качестве целого, а J - n-битной строкой, интерпретируемой как указанное подмножество от {1, . . . , n}. Инвертирование fss на выходе $(x_1, \ldots, x_n, y)$ требует нахождения подмножества $J' \in \{1, \ldots, n\}$, так чтобы $\sum_{j \in J'} x_j = y \bmod 2$. Студенты, которые изучали N P-полноту, могут напомнить, что эта проблема N P-полная. Но даже P *f*= N P не означает, что fss является односторонней: P *f*= N P означает, что каждый полиномиально-временной алгоритм не может решить проблему суммы подмножеств на, по меньшей мере, одном входе, поскольку для fss должна существовать односторонняя функция, необходимо, чтобы каждый полиномиально-временной алгоритм не мог решить проблему суммы подмножеств (по крайней мере, для определенных параметров) почти всегда. Таким образом, мы считаем, что описанная выше функция - это односторонняя функция, основанная на отсутствии известных алгоритмов, чтобы решить данную проблему даже при «небольшой» вероятности на случайных входах, а не только на основании того факта, что проблема является N P-полной.

Показывая семейство перестановок, мы пришли к выводу, что они, как полагают, являются односторонними. Пусть Gen - это такой вероятностный полиномиально-временной алгоритм, что когда на входе 1n, выводит n-битное простое число p наряду со специальным элементом $g \in \{2, \ldots, p-1\}$. (Элемент g должен быть генератором Z^*; см. раздел 8.3.3). Пусть Samp - это алгоритм который, при заданных p и g, выводит однородное целое число $x \in \{1, \ldots, p-1\}$. И, наконец, определим

$f_{p,g}(x) = [g^x \bmod p]$.

(Тот факт, что $f_{p,g}$ могут быть вычислены эффективно, следует из результатов в Приложении B.2.3). Можно показать, что эта функция является взаимно однозначной и, следовательно, перестановкой. Предполагаемая трудность инвертирования этой функции основана на сложности проблемы дискретного логарифмирования; мы скажем об этом намного больше в разделе 8.3.

Наконец, отметим, что очень эффективные односторонние функции могут быть получены из практических криптографических конструкций, таких как SHA-1 или AES в предположении, что они устойчивы к коллизиям или псевдослучайной перестановке, соответственно; см упражнения 7.4 и 7.5. (С технической точки зрения, они не могут соответствовать определению односторонности, по-

скольку имеют вход/выход фиксированной длины и, таким образом, мы не можем рассматривать их асимптотическое поведение. Тем не менее, вполне возможно предположить, что они являются односторонними в конкретном понимании).

Трудные предикаты

По определению, односторонняя функция сложна для инвертирования. Иначе говоря: с учетом $y = f(x)$ значение x невозможно вычислить в его полном объеме по любому полиномиально-временному алгоритму (кроме как, при пренебрежимо малой вероятности; мы это здесь игнорируем). Может сложиться впечатление, что ничего о x невозможно определить из $f(x)$ в течение полиномиального времени. Это не обязательно так. На самом деле для $f(x)$ возможна «утечка» немалого объема информации о x , даже если f является односторонней функцией. Для тривиального примера пусть g - односторонняя функция $f(x_1, x_2) \stackrel{\text{def}}{=} (x_1, g(x_2))$, где $|x_1| = |x_2|$. Легко показать, что f является односторонней функцией (это останется в качестве упражнения), даже если она показывает половину своего входа.

Для наших приложений нам нужно будет определить конкретную часть информации о x, которую «скрывает» $f(x)$. Это служит причиной появления понятия трудных предикатов. Трудный предикат $hc: \{0, 1\}^* \rightarrow \{0, 1\}$ функции f имеет свойство, которое $hc(x)$ трудно вычислить с вероятностью существенно выше, чем 1/2 с учетом $f(x)$. (Поскольку hc является булевой функцией, всегда возможно вычислить $hc(x)$ с вероятностью 1/2 при случайном прогнозе). Формально:

ОПРЕДЕЛЕНИЕ 7.4 *Функция $hc:\{0, 1\}^* \rightarrow \{0, 1\}$ является трудным предикатом функции f, если hc можно вычислить в течение полиномиального времени, и для любого вероятностного полиномиально-временного алгоритма A существует пренебрежимо малая функция negl , такая что*

$$\Pr_{x \leftarrow \{0,1\}^n}[\mathcal{A}(1^n, f(x)) = \mathsf{hc}(x)] \leq \frac{1}{2} + \mathsf{negl}(n)$$

где вероятность берется по равномерному выбору x in $\{0, 1\}^n$ и случайному выбору A.

Мы подчеркиваем, что $hc(x)$ является эффективно вычисляемой с учетом x (поскольку функция hc может быть рассчитана в течение полиномиального времени); определение требует, чтобы $hc(x)$ было сложно вычислить с учетом $f(x)$. Приведенное выше определение не требует, чтобы функция f была односторонней; если f - перестановка, то она, однако, не может быть трудным предикатом, если она не является односторонней (см. упражнение 7.13).

Простые идеи не работают. Рассмотрим логическое условие $\mathsf{hc}(x) \stackrel{\text{def}}{=} \bigoplus_{i=1}^{n} x_i$ где x1, . . . , xn обозначает биты x. Можно было бы надеяться, что это - трудный предикат какой-то односторонней функции f : если f невозможно инвертиро-

вать, то f (x) должна скрывать, по меньшей мере, один из битов xi своего прообраза x, который, по-видимому, подразумевает, что исключающее ИЛИ для всех битов x сложно вычислить. Несмотря на свою привлекательность, этот аргумент неверен. Чтобы увидеть это, допустим, что g - это односторонняя функция и определим $f(x) \stackrel{\text{def}}{=} (g(x), \bigoplus_{i=1}^{n} x_i)$. Нетрудно показать, что f является односторонней. Тем не менее очевидно, что f (x) не скрывает значение $\mathsf{hc}(x) = \bigoplus_{i=1}^{n} x_i$ потому, что это часть его выхода; следовательно, hc(x) не является трудным предикатом f . Распространив это, можно показать, что для любого фиксированного предиката hc имеется односторонняя функция f, для которой hc не является трудным предикатом f .

Тривиальные трудные предикаты. Некоторые функции имеют «тривиальные» трудные предикаты. Например, пусть f - функция, которая сбрасывает последний бит своего входа (i.e., $f(x_1 \cdots x_n) = x_1 \cdots x_{n-1}$). Трудно определить xn с учетом f (x) поскольку xn не зависит от выхода; таким образом, hc(x) = x^n является трудным предикатом f . Однако f не является односторонней. Когда мы используем трудные предикаты для наших конструкций, становится ясно, почему тривиальные трудные предикаты такого рода бесполезны.

От односторонних функций к псевдослучайности

Цель настоящей главы заключается в том, чтобы показать, как конструировать псевдослучайные генераторы, функции и перестановки на основе односторонней функции/перестановки. В этом разделе мы дадим обзор этих конструкций. Подробности приведены в следующих разделах.

Трудный предикат из какой-либо односторонней функции. Первый этап существует для того, чтобы показать, что трудный предикат существует для односторонней функции. На самом деле, вопрос истинности этого утверждения остается открытым; мы покажем что-то менее строгое, но достаточное для наших целей. А именно, мы покажем, что задав одностороннюю функцию f можно построить другую одностороннюю функцию g, вместе с трудным предикатом g.

ТЕОРЕМА 7.5 (*теорема Голдрейха - Левина (Goldreich–Levin)) Предположим, что существуют односторонние функции (соотв., перестановки). Тогда существует односторонняя функция (соотв., перестановка) g и трудный предикат hc функции g.*

Пусть f - односторонняя функция. Функции g и hc строятся следующим образом: зададим g(x, r) $\stackrel{\text{def}}{=}$ (f (x), r), для |x| = |r|, и определим

$$\mathsf{hc}(x, r) \stackrel{\text{def}}{=} \bigoplus_{i=1}^{n} x_i \cdot r_i,$$

где x_i (resp., ri)обозначает i-й бит x (соотв., r). Обратите внимание на то, что если r является равномерным, то hc(x, r) выходы исключающего ИЛИ случайного подмножества битов x. (Если ri = 1, то бит x^i исключается в XOR, и в противо-

ложном случае нет). Теорема Голдрейха-Левина, по существу, утверждает, что если f является односторонней функцией, то f (x) скрывает исключающее ИЛИ случайного подмножества битов x.

Псевдослучайные генераторы из односторонних перестановок. Следующий шаг должен показать, как трудный предикат односторонней перестановки может быть использован для построения псевдослучайного генератора. (Известно, что трудного предиката односторонней функции достаточно, но доказательство этого чрезвычайно сложное, и выходит за рамки данной книги). В частности, мы покажем:

ТЕОРЕМА 7.6 *Пусть f- односторонняя перестановка и hc - трудный предикат f . Тогда, tt(s)* $\stackrel{\text{def}}{=}$ *f (s) « hc(s) является псевдослучайным генератором с коэффициентом расширения A(n) = n + 1.*

Интуитивно догадываясь, для чего tt, как определено в теореме, образует псевдослучайный генератор, сначала заметим, что начальные n битов выхода tt(s) (т.е., битов f (s)) на самом деле распределены равномерно, если s распределено равномерно в силу того, что f является перестановкой. Далее, тот факт, что hc является трудным предикатом f означает, что hc(s) «выглядит случайной», т.е., является псевдо- случайной, даже с учетом f (s) (предполагая еще раз, что s является равномерным). Сопоставляя эти наблюдения, мы видим, что полный выход tt является псевдослучайным.

Псевдослучайные генераторы с произвольным расширением. Существование псевдослучайного генератора, который распространяет свое начальное число посредством даже единственного бита (как мы только что видели), уже весьма нетривиально. Но для приложений (например, для эффективного шифрования больших сообщений, как в разделе 3.3) нам требуется псевдослучайный генератор с гораздо большим расширением. К счастью, мы можем получить любой коэффициент полиномиального расширения, какой захотим:

ТЕОРЕМА 7.7 *Если существует псевдослучайный генератор с коэффициентом расширения A(n)=n+1, то для любого многочлена poly существует псевдослучайный генератор с коэффициентом расширения poly(n).*

Мы пришли к выводу, что псевдослучайные генераторы с произвольным (полиномиальным) расширением могут быть построены из любой односторонней перестановки.

Псевдослучайные функции/перестановки из псевдослучайных генераторов. Псевдослучайные генераторы, достаточные для построения безопасных с точки зрения EAV (расширенной проверки приложения) схем шифрования с закрытым ключом. Для достижения безопасного шифрования с закрытым ключом, с точки зрения атаки выбранного открытого текста (CPA) (не говоря уже о кодах аутентификации сообщений), тем не менее, мы опираемся на псевдослучайные функции. Следующий результат показывает, что второе можно построить из первого:

ТЕОРЕМА 7.8 *Если существует псевдослучайный генератор с коэффициентом расширения $A(n) = 2^n$, то существует псевдослучайная функция.*

На самом деле, мы можем сделать еще больше:

ТЕОРЕМА 7.9 *Если существует псевдослучайная функция, то существует и строгая псевдослучайная перестановка.*

Комбинируя все указанные выше теоремы, а также результаты глав 3 и 4, имеем следующие выводы:

СЛЕДСТВИЕ 7.10 Если предположить существование односторонних перестановок, то существуют псевдослучайные генераторы с любым полиномиальным коэффициентом расширения, псевдослучайные функции и строгие псевдослучайные перестановки.

СЛЕДСТВИЕ 7.11 Если предположить существование односторонних перестановок, то существуют схемы шифрования безопасные, с точки зрения атаки, на основе подобранного шифрованного текста и аутентификационные коды безопасных сообщений.

Как было отмечено ранее, все эти результаты можно получить только на основе существования односторонних функций.

Трудные предикаты из односторонних функций

В этом разделе мы докажем теорему 7.5, показав следующее:

ТЕОРЕМА 7.12 *Пусть f - односторонняя функция, определим g как* $g(x, r) \stackrel{\text{def}}{=} (f(x), r)$, *где* $|x| = |r|$. *Определим* $gl(x, r) \stackrel{\text{def}}{=} \oplus L^n_{i=1} x_i \bullet r_i$, *где* $x = x_1 \bullet\bullet\bullet x_n$ *и* $r = r_1 \bullet\bullet\bullet r_n$. *Then gl является трудным предикатом g.*

Ввиду сложности доказательства , мы докажем последовательно три более строгих результата, достигающие высшей точки в том, что утверждается в теореме.

Простой случай

Покажем сначала, что если существует полиномиально-временной оппонент A, который всегда правильно вычисляет gl(x, r) при условии g(x, r) = (f (x), r), то можно инвертировать f в течение полиномиального времени. С учетом предположения, что f является односторонней функций, следует, что такого противника A не существует.

ПРЕДПОЛОЖЕНИЕ 7.13 *Пусть f и gl такие, как в теореме 7.12. Если существует полиномиально-временной алгоритм A, такой что* $A(f(x), r) = gl(x, r)$ *для всех n и всех* $x, r \in \{0,1\}^n$, *то существует полиномиально-временной алгоритм Ar, такой что* $Ar(1^n, f(x)) = x$ *для всех n и всех* $x \in \{0,1\}^n$.

ДОКАЗАТЕЛЬСТВО Построим Ar следующим образом. $A^r(1^n, y)$ вычисляет $x_i := A(y, e_i)$ для i = 1, . . . , n, где ei обозначает n-битную строку с 1

в i-й позиции и 0 везде, кроме данной точки.. Тогда A^r выводит $x = x_1 \bullet\bullet\bullet x_n$. Очевидно, что A_r работает в течение полиномиального времени.

При выполнении $A^r(1^n, f(x\hat{}))$, значение x_i, рассчитанное A^r, удовлетворяет

$$x_i = \mathcal{A}(f(\hat{x}), e^i) = \mathrm{gl}(\hat{x}, e^i) = \bigoplus_{j=1}^{n} \hat{x}_j \cdot e^i_j = \hat{x}_i$$

Таким образом, $x_i = x\hat{}_i$ для всех i и тем самым A^r выводит правильную инверсию $x = x\hat{}$.

Если f – односторонняя, то для любого вероятностного алгоритма невозможно инвертировать f с пренебрежимо малой вероятностью. Таким образом, мы приходим к выводу, что не существует полиномиально-временного алгоритма, который всегда правильно вычисляет gl(x, r) из (f (x), r). Это довольно слабый результат, что очень далено от нашей цели показать, что gl(x, r) невозможно вычислить (с вероятностью существенно выше, чем 1/2) с учетом (f (x), r).

Более сложный случай

Теперь покажем, что для любого вероятностного полиномиально-временного алгоритма A сложно вычислить gl(x, r) из (f (x), r) с вероятностью существенно выше, чем 3/4. Мы снова покажем, что любой такой A предполагал бы существование полиномиально-временного алгоритма Ar , который инвертирует f с не пренебрежимо малой вероятностью. Обратите внимание, что стратегия доказательства предположения 7.13 терпит здесь неудачу, возможно, потому что A никогда не достигает цели, если r = ei (несмотря на то, что он может сработать успешно, скажем, при всех других значениях r). Кроме того, в данном случае A^r не знает, результат A(f (x), r) равен gl(x, r) или нет; единственное, что знает A^r – это то, что с большой вероятностью алгоритм A является правильным. Это еще более усложняет доказательство.

ПРЕДПОЛОЖЕНИЕ 7.14 *Пусть f и gl будут как в теореме 7.12. Если существует вероятностный полиномиально-временной алгоритм A и полином p(•), такой что*

$$\Pr_{x,r \leftarrow \{0,1\}^n} \left[\mathcal{A}(f(x), r) = \mathrm{gl}(x, r) \right] \geq \frac{3}{4} + \frac{1}{p(n)}$$

для бесконечного множества значений n, то существует вероятностный полиномиально-временной алгоритм Ar, такой что

$$\Pr_{x \leftarrow \{0,1\}^n} \left[\mathcal{A}'(1^n, f(x)) \in f^{-1}(f(x)) \right] \geq \frac{1}{4 \cdot p(n)}$$

для бесконечного множества значений n.

ДОКАЗАТЕЛЬСТВО Главное наблюдение, лежащее в основе доказательства этого предположения, заключается в том, что для каждого $r \in \{0, 1\}^n$, зна-

чения gl(x, r ⊕ e^i) и gl(x, r) вместе можно использовать, чтобы вычислить i-й бит x. (Напомним, что e^i обозначает n-битную строку с нулями везде, кроме i-й позиции.) Это справедливо, поскольку

$$\begin{aligned}&\mathrm{gl}(x,r)\oplus\mathrm{gl}(x,r\oplus e^i)\\&=\left(\bigoplus_{j=1}^{n}x_j\cdot r_j\right)\oplus\left(\bigoplus_{j=1}^{n}x_j\cdot(r_j\oplus e^i_j)\right)=x_i\cdot r_i\oplus\left(x_i\cdot\bar{r}_i\right)=x_i\end{aligned}$$

где $\bar{r}i$ – дополнение ri, а второе равенство является следствием того факта, что для j *f*= i значение xj • rj появляется в обеих суммах и, таким образом, сокращается. Приведенное выше показывает, что если A реагирует правильно как на (f (x), r) так и на (f (x), r ⊕ e^i), то Ar то можно правильно вычислить xi. К сожалению, A^r не знает, когда A реагирует правильно и когда нет; A^r знает только, что A реагирует правильно с «высокой» вероятностью. По этой причине A^r применит многочисленные случайные значения r, используя каждое из них, чтобы получить оценку x_i, а затем возьмет оценку, появляющуюся большую часть времени, в качестве ее итогового прогноза для x_i.

В качестве предварительного шага, мы покажем, что для многих x вероятность того, что A реагирует корректно, как для (f (x), r) так и (f (x), r ⊕ e^i), когда r однородное, достаточно высока. Это позволяет нам зафиксировать x и затем сосредоточиться на равномерном выборе r, который упрощает анализ.

УТВЕРЖДЕНИЕ 7.15 Пусть n будет таким, чтобы

$$\Pr_{x,r\leftarrow\{0,1\}^n}\left[\mathcal{A}(f(x),r)=\mathrm{gl}(x,r)\right]\geq\frac{3}{4}+\frac{1}{p(n)}$$

Тогда существует множество $S_n \in \{0, 1\}^n$ размера, минимум, 1/2p(n) , такое что для каждого $x \in S_n$ справедливо, что

$$\Pr_{r\leftarrow\{0,1\}^n}[\mathcal{A}(f(x),r)=\mathrm{gl}(x,r)]\geq\frac{3}{4}+\frac{1}{2p(n)}$$

ДОКАЗАТЕЛЬСТВО Пусть ε(n) = 1/p(n), определим $S_n \in \{0, 1\}^n$ должно быть множеством всех x^rs, для которых

$$\Pr_{r\leftarrow\{0,1\}^n}[\mathcal{A}(f(x),r)=\mathrm{gl}(x,r)]\geq\frac{3}{4}+\frac{\varepsilon(n)}{2}.$$

Мы имеем

$$\begin{aligned}\Pr_{x,r\leftarrow\{0,1\}^n}\left[\mathcal{A}(f(x),r)=\mathrm{gl}(x,r)\right]&=\frac{1}{2^n}\sum_{x\in\{0,1\}^n}\Pr_{r\leftarrow\{0,1\}^n}\left[\mathcal{A}(f(x),r)=\mathrm{gl}(x,r)\right]\\&=\frac{1}{2^n}\sum_{x\in S_n}\Pr_{r\leftarrow\{0,1\}^n}\left[\mathcal{A}(f(x),r)=\mathrm{gl}(x,r)\right]\\&\quad+\frac{1}{2^n}\sum_{x\notin S_n}\Pr_{r\leftarrow\{0,1\}^n}\left[\mathcal{A}(f(x),r)=\mathrm{gl}(x,r)\right]\end{aligned}$$

$$\leq \frac{|S_n|}{2^n} + \frac{1}{2^n} \cdot \sum_{x \notin S_n} \left(\frac{3}{4} + \frac{\varepsilon(n)}{2}\right)$$
$$\leq \frac{|S_n|}{2^n} + \left(\frac{3}{4} + \frac{\varepsilon(n)}{2}\right).$$

Поскольку $\frac{3}{4} + \varepsilon(n) \leq \Pr_{x,r \leftarrow \{0,1\}^n}\left[\mathcal{A}(f(x), r) = \mathsf{gl}(x, r)\right]$, простая алгебра $|S_n| \geq \frac{\varepsilon(n)}{2} \cdot 2^n$

Теперь дальнейшее логически вытекает как простое следствие.

УТВЕРЖДЕНИЕ 7.16 *Пусть n будет таким, чтобы*

$$\Pr_{x,r \leftarrow \{0,1\}^n}\left[\mathcal{A}(f(x), r) = \mathsf{gl}(x, r)\right] \geq \frac{3}{4} + \frac{1}{p(n)}$$

Тогда существует множество $S_n \in \{0, 1\}^n$ размера, минимум, 1/2p(n) , такое что для каждого x ∈Sn и каждого i справедливо следующее

$$\Pr_{r \leftarrow \{0,1\}^n}\left[\mathcal{A}(f(x), r) = \mathsf{gl}(x, r) \bigwedge \mathcal{A}(f(x), r \oplus e^i) = \mathsf{gl}(x, r \oplus e^i)\right] \geq \frac{1}{2} + \frac{1}{p(n)}$$

ДОКАЗАТЕЛЬСТВО Пусть ε(n) = 1/p(n), и примем, что S_n будет множеством, гарантированным предыдущим утверждением.. Нам известно, что для любого x ∈ S_n имеется

$$\Pr_{r \leftarrow \{0,1\}^n}[\mathcal{A}(f(x), r) \neq \mathsf{gl}(x, r)] \leq \frac{1}{4} - \frac{\varepsilon(n)}{2}$$

Зафиксируем i ∈ {1, …, n}. Если r распределено равномерно, то r ⊕ e^i; таким образом

$$\Pr_{r \leftarrow \{0,1\}^n}[\mathcal{A}(f(x), r \oplus e^i) \neq \mathsf{gl}(x, r \oplus e^i)] \leq \frac{1}{4} - \frac{\varepsilon(n)}{2}$$

Мы заинтересованы в более низкой ограничивающей вероятности, что A выводит правильный ответ как для gl(x, r), так и для gl(x, r ⊕ e^i); что то же самое, мы хотим для верхней границы вероятности, при которой A не удается вывести правильный ответ в любом из этих случаев. Заметим, что r и r ⊕ e^i не являются независимыми, таким образом, мы не можем просто перемножить вероятности неудачных ответов. Однако, мы можем применить границу объединения (см. предположение A.7) и сложить вероятности неудачных ответов. То есть, вероятность того, что A является неправильным или на gl(x, r) или gl(x, r ⊕ e^i) равна, не более чем

$$\left(\frac{1}{4} - \frac{\varepsilon(n)}{2}\right) + \left(\frac{1}{4} - \frac{\varepsilon(n)}{2}\right) = \frac{1}{2} - \varepsilon(n),$$

и, таким образом, A корректно как на gl(x, r), так и на gl(x, r ⊕ e^i) с вероятностью, не менее, чем 1/2 + ε(n). Это доказывает утверждение.

Для остальной части доказательства зададим ε(n) = 1/p(n) и рассмотрим толь-

ко те значения n, для которых

$$\Pr_{x,r\leftarrow\{0,1\}^n}\left[\mathcal{A}(f(x),r)=\mathrm{gl}(x,r)\right]\geq\frac{3}{4}+\varepsilon(n)\,. \qquad (7.1)$$

Предыдущее утверждение гласит, что для ε(n)/2 доля входов x, и любой i, алгоритм A реагирует правильно как на (f (x), r), так и на (f (x), r ⊕ e^i) с вероятностью не менее 1/2 + ε(n) при однородном выборе r, и с этого момента мы сосредоточимся только на таких значениях x. Построим вероятностный полиномиально-временной алгоритм Ar , который инвертирует f (x) с вероятностью 1/2, когда x ∈ Sn. Этого достаточно, чтобы затем доказать предположение 7.14 для бесконечного множества значений n,

$$\begin{aligned}&\Pr_{x\leftarrow\{0,1\}^n}[\mathcal{A}'(1^n,f(x))\in f^{-1}(f(x))]\\&\quad\geq\Pr_{x\leftarrow\{0,1\}^n}[\mathcal{A}'(1^n,f(x))\in f^{-1}(f(x))\mid x\in S_n]\cdot\Pr_{x\leftarrow\{0,1\}^n}[x\in S_n]\\&\quad\geq\frac{1}{2}\cdot\frac{\varepsilon(n)}{2}\;=\;\frac{1}{4p(n)}\,.\end{aligned}$$

Алгоритм A^r, заданный как вход 1^n и y, работает следующим образом:

1. Для i = 1, . . . , n выполним:

• Несколько раз выберем однородное r ∈ $\{0, 1\}^n$ и вычисдим A(y, r) ⊕ A(y, r ⊕ e^i) в качестве «оценки» i-го бита прообраза y. После выполнения этой процедуры достаточно много раз (см. ниже), будем считать xi «оценкой» того, что происходит большую часть времени.

2. Выведем x = x_1 • • • x_n.

Наметим анализ вероятности того, что A^r правильно инвертирует заданный вход y. (Мы позволим себе быть немного лаконичнее, поскольку полное доказательство для более трудного случая приведено в следующем разделе). Пусть y = f (x^) и напомним, что мы предполагаем здесь, что n такое, что уравнение (7.1) справедливо и x^ ∈ S_n. Зафиксируем некоторое i. Предыдущее требование предполагает, что оценка A(y, r) ⊕ A(y, r ⊕ e^i) равна gl(x^, e^i) с вероятностью не менее 1 + ε(n) по выбору r. Получая достаточно много оценок и позволяя xi быть мажоритарным значением, Ar можно гарантировать, что xi равно gl(x^, e^i) с вероятностью не менее 1 − 1 . Конечно, нужно убедиться, что полиномиально множество оценок достаточное. К счастью, поскольку ε(n) = 1/p(n) для некоторого полинома p и независимое значение r используется для получения каждой оценки, граница Чернофф (Chernoff) (ср. с предположением A.14) показывает, что полиномиально множества оценок достаточно.

Подводя итог, имеем, что для каждого i значение x^i, вычисленное Ar , некорректно с вероятностью не более 1/2n. Граница объединения, таким образом,

показывает, что Ar является некорректным для некоторого i с вероятностью не более n •1/2n= /2. То есть, A^r является правильным для всех i—и, таким образом правильно инвертирует y—с вероятностью не менее 1 − 1/2 = 1/2 . Это завершает доказательство предположения

Следствие предположения 7.14 заключается в том, что если f является односторонней функцией, то для любого полиномиально-временного алгоритма A вероятность того, что A правильно прогнозирует gl(x, r) при заданном (f (x), r), чаще всего, незначительно больше, чем 3/4.

Полное доказательство

Мы предполагаем знакомство с упрощенными доказательствами в предыдущих разделах и основываемся на идеях, развитых там. Мы основываемся на некоторой терминологии и стандартных результатах из теории вероятности, обсуждаемых в Приложении A.3. Докажем следующее предположение, которое выражает теорема 7.12:

ПРЕДПОЛОЖЕНИЕ 7.17 *Пусть f и gl будут как в теореме 7.12. Если существует вероятностный полиномиально-временной алгоритм A и полином p(•), такой что*

$$\Pr_{x,r\leftarrow\{0,1\}^n}\left[\mathcal{A}(f(x),r)=\mathrm{gl}(x,r)\right]\geq\frac{1}{2}+\frac{1}{p(n)}$$

для бесконечного множества значений n, то существует вероятностный полиномиально-временной алгоритм Ar и полином pr(•), такой что

$$\Pr_{x\leftarrow\{0,1\}^n}\left[\mathcal{A}'(1^n,f(x))\in f^{-1}(f(x))\right]\geq\frac{1}{p'(n)}$$

для бесконечного множества значений n.

ДОКАЗАТЕЛЬСТВО Еще раз зададим ε(n) = 1/p(n) и рассмотрим только те значения n, для которых $\Pr_{x,r\leftarrow\{0,1\}^n}\left[\mathcal{A}(f(x),r)=\mathrm{gl}(x,r)\right]\geq\frac{1}{2}+\frac{1}{p(n)}$. Дальнейшее аналогично утверждению 7.15 и доказывается тем же способом.

УТВЕРЖДЕНИЕ 7.18 Пусть n будет таким, чтобы

$$\Pr_{x,r\leftarrow\{0,1\}^n}\left[\mathcal{A}(f(x),r)=\mathrm{gl}(x,r)\right]\geq\frac{1}{2}+\varepsilon(n).$$

Тогда существует множество $S_n \in \{0, 1\}^n$ размера не менее ε(n)/2 • 2^n , такое что для каждого x ∈ S_n справедливо, что

$$\Pr_{r\leftarrow\{0,1\}^n}[\mathcal{A}(f(x),r)=\mathrm{gl}(x,r)]\geq\frac{1}{2}+\frac{\varepsilon(n)}{2}. \qquad (7.2)$$

Если мы начнем, пытаясь доказать аналогию утверждению 7.16, то лучшее, что мы может утверждать здесь это то, что если x ∈ Sn , то мы имеем

$$\Pr_{r\leftarrow\{0,1\}^n}\left[\mathcal{A}(f(x),r)=\mathrm{gl}(x,r)\bigwedge\mathcal{A}(f(x),r\oplus e^i)=\mathrm{gl}(x,r\oplus e^i)\right]\geq\varepsilon(n)$$

для любого i. Таким образом, если мы пытаемся использовать A(f (x), r) ⊕ A(f (x), r ⊕ e^i) в качестве оценки для xi, то все, что мы можем утверждать - это то, что эта оценка будет корректной с вероятностью не менее ε(n), которая не может быть лучше, чем принять случайный прогноз! Мы не можем утверждать, что инверсия результата дает хорошую оценку в любом случае.

Вместо этого, мы разработаем A^r так, чтобы он вычислял gl(x, r) и gl(x, r⊕e^i) при вызове A только один раз. Мы сделаем это, имея A^r, запустим A(x, r ⊕ e^i), и имея A^r просто "угадаем" само значение gl(x, r). Наивным способом сделать это, было бы выбрать независимые друг от друга r, как и ранее, и имея A^r, дать независимый прогноз gl(x, r) для каждого значения r. Но тогда вероятность того, что все эти прогнозы правильные , что, как мы увидим, необходимо, если A^r является выходом корректной инверсии, было бы пренебрежимо малым, поскольку используется полиномиальное множество r.

Ключевое наблюдение настоящего доказательства состоит в том, что A^r может генерировать r парно-независимым способом, и делать свои прогнозы определенным образом так, чтобы его прогнозы были правильными, с не пренебрежимо малой вероятностью. В частности, чтобы генерировать m значений r, мы имеем Ar , выбираем A = |log(m + 1)| независимо и равномерно распределенные строки s1, . . . , sA ∈ $\{0,1\}^n$. Затем, для каждого непустого подмножества I∈{1, . . . , A}, мы задаем $r_I := \oplus_{i\ I}\, s_i$. Поскольку существуют $2^A - 1$ непустых подмножеств, это определяет набор $2^{\lfloor\log(m+1)\rfloor} - 1 \geq m$ строк. Каждая такая строка имеет равномерное распределение. Эти строки не являются независимыми, но они являются попарно независимыми. Чтобы убедиться в этом, обратите внимание, что для каждых двух подмножеств I ≠ J существует индекс j ∈ I∩ J, такой что j ∉/ I ∩ J . Без потери общности, предположим j ∈ I∩ I. Тогда значение sj является равномерным и независимым от значения r_I . Поскольку sj включено в исключающее ИЛИ (XOR), которое определяет r_J , это означает, что r_J также является равномерным и независимым от r_I .

Теперь у нас есть два следующих важных замечания:

1. С учетом gl(x, s^1), . . . , gl(x, s^A), можно вычислить gl(x, r^I) для каждого подмножества I ∈ {1, . . . , A}. Это потому, что gl(x, r^I) = gl(x, ⊕i I s^1) = ⊕gl(x, s^1).

2. Если A^r просто прогнозирует значения gl(x, s^1), . . . , gl(x, s^A) выбирая однородный бит для каждого, то все эти прогнозы будут корректными с вероятностью $1/2^A$. Если m является полиномом в параметре безопасности n, то $1/2^A$ не пренебрежимо мало, и, таким образом, при не пренебрежимо малой вероятности A^r корректно прогнозирует все значения gl(x, s^1), . . . , gl(x, s^A).

Сочетание указанного выше дает способ получения m = poly(n) равномерных и попарно-независимых строк {r_I }, наряду с корректными значениями для {gl(x, r_I)} при не

пренебрежимо малой вероятности.. Эти значения затем можно использовать, чтобы вычислить xi тем же способом, как и в доказательстве предположения 7.14. Детали далее.

Алгоритм инверсии A^r. Теперь мы дадим полное описание алгоритма Ar, который принимает входные данные 1^n, y и пытается вычислить инверсию y. Алгоритм действует следующим образом:

1. Зададим $A := |\log(2n/\varepsilon(n)^2 + 1)|$.

2. Выберем равномерные, независимые $s^1, \ldots, s^A \in \{0,1\}^n$ и $\sigma^1, \ldots, \sigma^A \in \{0,1\}$.

3. Для любого непустого подмножества $I \in \{1, \ldots, A\}$, вычислим $r^I := \oplus_{i \in I} s^i$ и $\sigma I := \oplus_{i \in I} \sigma^i$.

4. Для i = 1, . . . , n выполним:

(a) Для каждого непустого подмножества $I \in \{1, \ldots, A\}$, зададим

$$x_i^I := \sigma^I \oplus \mathcal{A}(y, r^I \oplus e^i)$$

(b) Зададим $x_i := \text{majority}_I \{x^I\}$ (т.е., берем бит, который появляется большую часть времени на предыдущем этапе).

5. Выводим x = x1 • • • xn.

Остается вычислить вероятность того, что Ar выводит $x \in f^{-1}(y)$. Как и в доказательстве предположения 7.14, сосредоточимся только на n как в утверждении 7.18 и предположим, что y = f(x^) для некоторого x^∈ Sn. Каждое σi представляет собой «прогноз» для значения gl(x^, si). Как отмечалось ранее, с не пренебрежимо малой вероятностью все эти прогнозы являются корректными; покажем, что обусловленный этим результатом Ar выводит x = x^ с вероятностью не менее 1/2.

Предположим, что $\sigma i = gl(x^, s^i)$ для всех i. Тогда $\sigma I = gl(x^, r^I)$ для всех I. Зафиксируем индекс $i \in \{1, \ldots, n\}$ и рассмотрим вероятность того, что Ar получает корректное значение xi = x^i. Для любого непустого I имеем $A(y, r^I \oplus e^i) = gl(x^, r^I \oplus e^i)$ с вероятностью не менее $1 + \varepsilon(n)/2$ на основании выбора r; это следует из того, что $x^ \in S^n$ и $rI \oplus e^i$ распределены равномерно. Таким образом, для любого непустого подмножества I имеем $Pr[xI = x^i] \geq 1/2 + \varepsilon(n)/2$. Кроме того, {xI }I1,...,A являются попарно-независимыми, поскольку {rI }I∈{1,...,A}(и поэтому {rI ⊕ ei}I∈{1,...,A}) являются попарно, независимыми. Поскольку xi определяется как значение, которое появляется большую часть времени среди {xI }I1,...,A, можно применить предположение A.13, чтобы получить

$$\begin{aligned}
\Pr[x_i \neq \hat{x}_i] &\leq \frac{1}{4 \cdot (\varepsilon(n)/2)^2 \cdot (2^\ell - 1)} \\
&\leq \frac{1}{4 \cdot (\varepsilon(n)/2)^2 \cdot (2n/\varepsilon(n)^2)} \\
&= \frac{1}{2n}.
\end{aligned}$$

Указанное выше, справедливо для всех i, таким образом, мы можем видеть, что вероятность того, что xi f= x^i для some i составляет не более 1/2. То есть, xi = x^i для всех i (и, следовательно x = x^) с вероятностью не менее 1/2.

Сопоставим все изложенное: Пусть n будет как в утверждении 7.18 и y = f (x^). С вероятностью не менее ε(n)/2, имеем x^ ∈ S_n. Все эти прогнозы σi являются корректными с вероятностью не менее

$$\frac{1}{2^\ell} \geq \frac{1}{2 \cdot (2n/\varepsilon(n)^2 + 1)} > \frac{\varepsilon(n)^2}{5n}$$

для достаточно большого n. При выполнении обоих указанных выше условий, Ar выводит x = x^ с вероятностью не менее 1/2. Общая вероятность, при которой Ar инвертирует свои входные данные составляет, таким образом, не менее $\varepsilon(n)^3/20n = 1/(20\ np(n)^3)$ для бесконечного множества n.

Поскольку $20\ np(n)^3$ – полином от n, это доказывает предположение 7.17.

Построение псевдослучайных генераторов

Сначала покажем, как построить псевдослучайные генераторы, которые расширяют свои входные данные на один бит в предположениии, что существует односторонняя перестановка. Затем мы покажем, как расширить это, чтобы получить какой-либо полиномиальный коэффициент расширения.

Псевдослучайные генераторы с минимальным расширением

Пусть f – односторонняя перестанока с трудным предикатом hc. Это означает, что hc(s) «выглядит случайным» с учетом f (s), когда s является равномерным. Кроме того, так как f является перестановкой, то сама f (s) распределена равномерно. (Применение перестановки к равномерно распределенному значению, дает равномерно распределенное значение). Таким образом, если s является равномерной n-битной строкой, то (n +1)-битная строка f (s)‖hc(s) состоит из равномерной n-битной строки плюс дополнительный бит, который выглядит равномерным, даже обусловленный начальными n битами; другими словами, (n + 1)-битная строка является псевдослучайной. Таким образом, алгоритм tt, определяемый tt(s) = f (s)‖hc(s) представляет собой псевдослучайный генератор.

ТЕОРЕМА 7.19 *Пусть f – односторонняя перестановка с трудным предикатом hc. Тогда алгоритм tt, определяемый tt(s) = f (s)||hc(s), является псевдослучайным генератором с коэффициентом расширения A(n) = n + 1.*

ДОКАЗАТЕЛЬСТВО Пусть D – вероятностный полиномиально-временной алгоритм. Докажем, что существует пренебрежимо малая функция negl, такая что

$$\Pr_{r \leftarrow \{0,1\}^{n+1}}[D(r) = 1] - \Pr_{s \leftarrow \{0,1\}^n}[D(G(s)) = 1] \leq \mathsf{negl}(n). \qquad (7.3)$$

Подобный аргумент показывает, что существует пренебрежимо малая функция $\mathrm{negl}^{\mathrm{r}}$, для которой

$$\Pr_{s\leftarrow\{0,1\}^n}[D(G(s))=1]-\Pr_{r\leftarrow\{0,1\}^{n+1}}[D(r)=1]\le \mathsf{negl}'(n),$$

что завершает доказательство.

Сначала отметим

$$\begin{aligned}\Pr_{r\leftarrow\{0,1\}^{n+1}}[D(r)=1] &= \Pr_{r\leftarrow\{0,1\}^n,\, r'\leftarrow\{0,1\}}[D\left(r\|r'\right)=1]\\ &= \Pr_{s\leftarrow\{0,1\}^n, r'\leftarrow\{0,1\}}[D\left(f(s)\|r'\right)=1]\\ &= \frac{1}{2}\cdot\Pr_{s\leftarrow\{0,1\}^n}[D\left(f(s)\|\mathsf{hc}(s)\right)=1]\\ &\quad+\frac{1}{2}\cdot\Pr_{s\leftarrow\{0,1\}^n}[D\left(f(s)\|\overline{\mathsf{hc}}(s)\right)=1],\end{aligned}$$

используя тот факт, что f является перестановкой для второго равенства, и того, что равномерный бит rr равен hc(s) с вероятностью точно 1/2 для третьего равенства. Поскольку

$$\Pr_{s\leftarrow\{0,1\}^n}[D(G(s))=1]=\Pr_{s\leftarrow\{0,1\}^n}[D\left(f(s)\|\mathsf{hc}(s)\right)=1]$$

(при определении tt), это означает, что равенство (7.3) эквивалентно

$$\frac{1}{2}\cdot\left(\Pr_{s\leftarrow\{0,1\}^n}[D\left(f(s)\|\overline{\mathsf{hc}}(s)\right)=1]-\Pr_{s\leftarrow\{0,1\}^n}[D\left(f(s)\|\mathsf{hc}(s)\right)=1]\right)\le\mathsf{negl}(n).$$

Рассмотрим следующий алгоритм A, который задает в качестве входного значение y = f (s) и пытается предсказать значение hc(s):

1. Выберем равномерное $\mathrm{r}^{\mathrm{r}}\in\{0,1\}$.

2. Запустим D(y‖r$^{\mathrm{r}}$). Если D выводит 0, выведем rr; иначе выведем $\mathrm{r}^{\overline{\mathrm{r}}}$. Очевидно, что A работает в течение полиномиального времени. По определению A, имеем

$$\begin{aligned}&\Pr_{s\leftarrow\{0,1\}^n}[\mathcal{A}(f(s))=\mathsf{hc}(s)]\\ &=\frac{1}{2}\cdot\Pr_{s\leftarrow\{0,1\}^n}[\mathcal{A}(f(s))=\mathsf{hc}(s)\mid r'=\mathsf{hc}(s)]\\ &\quad+\frac{1}{2}\cdot\Pr_{s\leftarrow\{0,1\}^n}[\mathcal{A}(f(s))=\mathsf{hc}(s)\mid r'\neq\mathsf{hc}(s)]\\ &=\frac{1}{2}\cdot\left(\Pr_{s\leftarrow\{0,1\}^n}[D(f(s)\|\mathsf{hc}(s))=0]+\Pr_{s\leftarrow\{0,1\}^n}[D(f(s)\|\overline{\mathsf{hc}}(s))=1]\right)\\ &=\frac{1}{2}\cdot\left(\left(1-\Pr_{s\leftarrow\{0,1\}^n}[D(f(s)\|\mathsf{hc}(s))=1]\right)+\Pr_{s\leftarrow\{0,1\}^n}[D(f(s)\|\overline{\mathsf{hc}}(s))=1]\right)\\ &=\frac{1}{2}+\frac{1}{2}\cdot\left(\Pr_{s\leftarrow\{0,1\}^n}[D(f(s)\|\overline{\mathsf{hc}}(s))=1]-\Pr_{s\leftarrow\{0,1\}^n}[D\left(f(s)\|\mathsf{hc}(s)\right)=1]\right).\end{aligned}$$

Поскольку hc является трудным предикатом f , то отсюда следует, что существует пренебрежимо малая функция negl, для которой

$$\frac{1}{2}\cdot\left(\Pr_{s\leftarrow\{0,1\}^n}[D\left(f(s)\|\overline{\mathsf{hc}}(s)\right)=1]-\Pr_{s\leftarrow\{0,1\}^n}[D\left(f(s)\|\mathsf{hc}(s)\right)=1]\right)\leq \mathsf{negl}(n),$$

что и требовалось доказать.

Увеличение коэффициента расширения

Теперь покажем, что коэффициент расширения псевдослучайного генератора может быть увеличен на любую требуемую (полиномиальную) величину. Это означает, что предыдущая конструкция с коэффициентом расширения A(n) = n + 1 достаточна для построения псевдослучайного генератора с произвольным (полиномиальным) коэффициентом расширения.

ТЕОРЕМА 7.20 *Если существует псевдослучайный генератор tt с коэффициентом расширения n + 1, то для любого полинома poly существует псевдослучайный генератор tt^ с коэффициентом расширения poly(n).*

ДОКАЗАТЕЛЬСТВО Сначала рассмотрим построение псевдослучайного генератора tt^ , который выводит n + 2 битов.tt^ работает следующим образом: При заданном начальном значении $s \in \{0, 1\}^n$ вычисляется t1 := tt(s), чтобы получить n + 1 псевдослучайных битов. Затем начальные n биты из t1 снова используются в качестве начального значения для tt; результирующие n +1 битов, последовательно соединенных с последним битом t1, что дает (n + 2)-битный выход. (См. рис. 7.1). Второе применение tt использует псевдослучайное начальное значение, а не случайное. Доказательство, которое мы приведем далее, показывает, что это не влияет на псевдослучайность вывода.

Теперь докажем, что tt^ является псевдослучайным генератором. Определим три последовательности распределений $\{H_0\}_{n=1,...}$, $\{H1\}_{n=1,...}$ и $\{H2\}_{n=1,...}$, где каждое из H0, и Hn является распределением строк длиной n + 2. В распределении Hn выбирается равномерная строка $t0 \in \{0, 1\}^n$, и выход – это tt^(t0). В распределении n, выбирается равномерная строка $t1 \in \{0, 1\}^{n+1}$ и анализируется как s1‖σ1 (где s1 –начальные n битов t_1 и σ_1 – конечный бит). Выход равен $t_2 := tt(s_1)\|\sigma_1$.

В распределении H_2, выход является равномерной строкой $t_2 \in \{0, 1\}^{n+2}$. Обозначим как $t_2 \leftarrow H_i$ процесс генерации (n + 2)-битной строки t_2 в соответствии с распределением H_i .

Зафиксируем произвольный вероятностный полиномиально-временной отличительный признак D. Сначала покажем, что существует пренебрежимо малая функция neglr, такая что

$$\left| \Pr_{t_2 \leftarrow H_n^0}[D(t_2) = 1] - \Pr_{t_2 \leftarrow H_n^1}[D(t_2) = 1] \right| \leq \mathsf{negl}'(n). \qquad (7.4)$$

Чтобы убедиться в этом, рассмотрим полиномиально-временной отличительный признак Dr , который на входе $t_1 \in \{0, 1\}^{n+1}$, анализирует t1 как s1||σ1 с |s1| = n, вычисляет $t_2 := tt(s_1)||\sigma_1$, и выводит $D(t_2)$. Очевидно, что Dr действует в течение полиномиального времени. Заметим, что:

1. Если t_1 –равномерно, то распределение по t2 , сгенерированое Dr точно такое, как распределение H^1_n. Таким образом,

$$\Pr_{t_1 \leftarrow \{0,1\}^{n+1}}[D'(t_1) = 1] = \Pr_{t_2 \leftarrow H_n^1}[D(t_2) = 1].$$

2. Если $t_1 = tt(s)$ для равномерного $s \in \{0, 1\}^n$, то распределение по t_2, сгенерированное Dr то же, что и в распределении H^0_n. То есть,

$$\Pr_{s \leftarrow \{0,1\}^n}[D'(G(s)) = 1] = \Pr_{t_2 \leftarrow H_n^0}[D(t_2) = 1].$$

Псевдослучайность tt предполагает, что существует пренебрежимо маля функция negl^r c

$$\left| \Pr_{s \leftarrow \{0,1\}^n}[D'(G(s)) = 1] - \Pr_{t_1 \leftarrow \{0,1\}^{n+1}}[D'(t_1) = 1] \right| \leq \mathsf{negl}'(n).$$

Выражение (7.4) соблюдается.Далее мы утверждаем, что существует пренебрежимо малая функция $\mathrm{negl}r^r$, такая что

$$\left| \Pr_{t_2 \leftarrow H_n^1}[D(t_2) = 1] - \Pr_{t_2 \leftarrow H_n^2}[D(t_2) = 1] \right| \leq \mathsf{negl}''(n). \qquad (7.5)$$

Чтобы убедиться в этом, рассмотрим полиномиально-временной отличительный признак D^{rr} , что на входе $w \in \{0, 1\}^{n+1}$, выбирает равномерное $\sigma 1 \in \{0, 1\}$, задает $t_2 := w||\sigma_1$ и выводит $D(t_2)$. Если w равномерное, то такое же и t_2; поэтому,

$$\Pr_{w \leftarrow \{0,1\}^{n+1}}[D''(w) = 1] = \Pr_{t_2 \leftarrow H_n^2}[D(t_2) = 1].$$

С другой стороны, если $w = tt(s)$ для равномерного $s \in \{0, 1\}^n$, то t_2 распределяется так же в соответствии с H_1 и, таким образом

$$\Pr_{s \leftarrow \{0,1\}^n}[D''(G(s)) = 1] = \Pr_{t_2 \leftarrow H_n^1}[D(t_2) = 1].$$

Как и ранее, псевдослучайность tt предполагает выражение (7.5).

Сложив все вместе, имеем

$$\left|\Pr_{s\leftarrow\{0,1\}^n}[D(\hat{G}(s))=1]-\Pr_{r\leftarrow\{0,1\}^{n+2}}[D(r)=1]\right| \tag{7.6}$$

$$=\left|\Pr_{t_2\leftarrow H_n^0}[D(t_2)=1]-\Pr_{t_2\leftarrow H_n^2}[D(t_2)=1]\right|$$

$$\leq\left|\Pr_{t_2\leftarrow H_n^0}[D(t_2)=1]-\Pr_{t_2\leftarrow H_n^1}[D(t_2)=1]\right|$$

$$+\left|\Pr_{t_2\leftarrow H_n^1}[D(t_2)=1]-\Pr_{t_2\leftarrow H_n^2}[D(t_2)=1]\right|$$

$$\leq \mathsf{negl}'(n)+\mathsf{negl}''(n),$$

используя выражения (7.4) и (7.5). Так как D был произвольным полиномиально-временным отличительным признаком, это доказывает, что ttˆ является псевдослучайным генератором.

Общий случай. Та же идея, что и выше, может быть итеративно применена, чтобы генерировать столько псевдослучайных битов, сколько необходимо. Говоря формально, мы хотим построить псевдослучайный генератор ttˆ с коэффициентом расширения n + p(n), для некого полинома p. На входе $s \in \{0, 1\}^n$ алгоритм ttˆ делает (см. рис. 7.1):

1. Зададим $t_0 := s$. Для $i = 1, \ldots, p(n)$ выполним:

(a) Пусть s_{i-1} будут первые n битов из t_{i-1}, и пусть σ_{i-1} обозначает остальные $i - 1$ биты. (Если $i = 1$, $s_0 = t_0$ и σ_0 является пустой строкой.)

(b) Зададим $t_i := tt(sr)\|\sigma_{i-1}$.

2. Выведем tp(n).

Покажем, что ttˆ – псевдослучайный генератор. Это доказательство использует общепринятую технику, известную как гибридный аргумент. (На самом деле, даже в случае p(n) = 2, приведенном выше, использовался гибридный аргумент). Основное отличие от предыдущего доказательства – техническое. Ранее мы могли бы определить и четко работать с тремя последовательностями распределений $\{H_n^0\}$, $\{H_n^1\}$, and $\{H_n^2\}$. В данном случае это невозможно, поскольку количество рассматриваемых распределений увеличивается с ростом n.

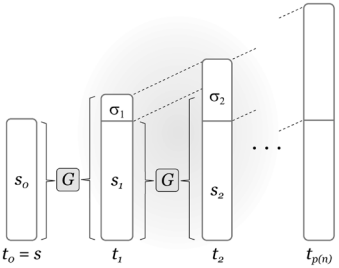

РИС. 7.1: Увеличение расширения псевдослучайного генератора.

Для любого n и $0 \le j \le p(n)$, пусть Hj будет распределением строк длиной n +p(n), определенных следующим образом: выберем равномерное $tj \in \{0,1\}^{n+j}$, затем запустим tt^ , начиная с итерации j + 1 и выведем tp(n). (Когда j = p(n), это означает, что мы просто выбираем равномерное $tp(n) \in \{0,1\}^{n+p(n)}$ и выводим его.) Ключевое наблюдение состоит в том, что H_0 соответствует выводимому tt^(s) для равномерного $s \in \{0,1\}^n$, тогда как соответствует выводимой равномерной (n + p(n))-битной строке. Зафиксируем любой полиномиально-временной отличительный признак D, это означает

$$\left| \Pr_{s \leftarrow \{0,1\}^n}[D(\hat{G}(s)) = 1] - \Pr_{r \leftarrow \{0,1\}^{n+p(n)}}[D(r) = 1] \right|$$

$$= \left| \Pr_{t \leftarrow H_n^0}[D(t) = 1] - \Pr_{t \leftarrow H_n^{p(n)}}[D(t) = 1] \right|. \qquad (7.7)$$

.

Мы доказали, что указанная выше функция пренебрежимо мала, следовательно tt^ является псевдослучайным генератором.

Зафиксируем D, как указано выше, и рассмотрим отличительный признак D^r, который делает следующее, когда в качестве входных данных задана строка $w \in \{0,1\}^{n+1}$:

1. Выберем равномерное $j \in \{1, \ldots, p(n)\}$.

2. Выберем равномерное $\sigma r \in \{0,1\}^{j-1}$. (Когда j = 1, тогда σr– пустая строка).

3. Зададим tj := w||σ^r. Затем запустим tt^, начиная с итерации j + 1, чтобы вычислить $tp(n) \in \{0,1\}^{n+p(n)}$. Выведем D(tp(n)).

Очевидно, что Dr действует в течение полиномиального времени. Анализируя поведение Dr, отметим, что оно является более сложным, чем ранее, хотя основные идеи те же. Зафиксируем n и, скажем, что Dr выбирает j = j*. Если w является равномерным, то tj* является равномерным и, таким образом, распределение по $t \stackrel{\text{def}}{=} t\,p(n)$ точно такое же, как и распределение H_j. То есть,

$$\Pr_{w \leftarrow \{0,1\}^{n+1}}[D'(w) = 1 \mid j = j^*] = \Pr_{t \leftarrow H_n^{j^*}}[D(t) = 1].$$

Поскольку каждое значение для j выбирается с равной вероятностью,

$$\Pr_{w \leftarrow \{0,1\}^{n+1}}[D'(w) = 1] = \frac{1}{p(n)} \cdot \sum_{j^*=1}^{p(n)} \Pr_{w \leftarrow \{0,1\}^{n+1}}[D'(w) = 1 \mid j = j^*]$$

$$= \frac{1}{p(n)} \cdot \sum_{j^*=1}^{p(n)} \Pr_{t \leftarrow H_n^{j^*}}[D(t) = 1]. \qquad (7.8)$$

С другой стороны, скажем Dr выбирает j = j* и w = tt(s) для равномерного s ∈ {0, 1}n. Определив $t_{j^*-1} = s \| \sigma^r$, мы видим, что t_{j^*-1} является равномерным и, таким образом, эксперимент с участием D^r эквивалентен запуску tt^ от итерации j*, чтобы вычислить tp(n). То есть распределение по $t \stackrel{\text{def}}{=} t_{p(n)}$ теперь точно такое же, как $H_n^{j^*-1}$. То есть,

$$\Pr_{s \leftarrow \{0,1\}^n}[D'(G(s)) = 1 \mid j = j^*] = \Pr_{t \leftarrow H_n^{j^*-1}}[D(t) = 1].$$

Следовательно,

$$\begin{aligned}
\Pr_{s \leftarrow \{0,1\}^n}[D'(\hat{G}(s)) = 1] &= \frac{1}{p(n)} \cdot \sum_{j^*=1}^{p(n)} \Pr_{s \leftarrow \{0,1\}^n}[D'(G(s)) = 1 \mid j = j^*] \\
&= \frac{1}{p(n)} \cdot \sum_{j^*=1}^{p(n)} \Pr_{t \leftarrow H_n^{j^*-1}}[D(t) = 1] \\
&= \frac{1}{p(n)} \cdot \sum_{j^*=0}^{p(n)-1} \Pr_{t \leftarrow H_n^{j^*}}[D(t) = 1]. \qquad (7.9)
\end{aligned}$$

Теперь мы можем проанализировать, насколько хорошо Dr отличает выходные данные tt от случайных:

$$\begin{aligned}
&\left| \Pr_{s \leftarrow \{0,1\}^n}[D'(G(s)) = 1] - \Pr_{w \leftarrow \{0,1\}^{n+1}}[D'(w) = 1] \right| \qquad (7.10) \\
&= \frac{1}{p(n)} \cdot \left| \sum_{j^*=0}^{p(n)-1} \Pr_{t \leftarrow H_n^{j^*}}[D(t) = 1] - \sum_{j^*=1}^{p(n)} \Pr_{t \leftarrow H_n^{j^*}}[D(t) = 1] \right| \\
&= \frac{1}{p(n)} \cdot \left| \Pr_{t \leftarrow H_n^0}[D(t) = 1] - \Pr_{t \leftarrow H_n^{p(n)}}[D(t) = 1] \right|,
\end{aligned}$$

опираясь на выражения (7.8) и (7.9) для первого равенства. (Второе равенство выполняется, поскольку в каждую сумму входят одни и те же элементы, за исключением первого элемента левой суммы и последнего элемента правой суммы). Поскольку tt является псевдослучайным генератором, то элемент с левой стороны равенства (7.10) – пренебрежимо мал; поскольку p является полиномом, это означает, что равенство (7.7) пренебрежимо мало, завершая доказательство, что tt^ – псевдослучайный генератор.

Соединим все это вместе. Пусть f будет односторонней перестановкой. Используя псевдослучайный генератор с коэффициентом расширения n + 1 из теоремы 7.19, и увеличивая коэффициент расширения до n + A, используя подход из доказательства теоремы 7.20, получим следующий псевдослучайный генератор tt^:

$$tt\hat{}(s) = f^{(A)}(s) \| hc(f^{(A-1)}(s)) \| \bullet\bullet\bullet \| hc(s),$$

где f (i)(s) относится к i-кратной итерации f . Обратите внимание, что tt^ использует l оценок f и генерирует один псевдослучайный бит на одну оценку,

используя трудный предикат hc.

Соединение с потоковыми шифрами. Напомним из раздела 3.3.1, что потоковый шифр (без IV) определяется с помощью алгоритмов (Init, GetBits), где Init берет начальное число $s \in \{0, 1\}^n$ и возвращает исходное состояние st, а GetBits использует в качестве входных данных текущее состояние st и выводит бит σ, а также обновляет состояние str. Конструкция tt^из предыдущего доказательства вполне вписывается в эту парадигму: возьмем Init в качестве тривиального алгоритма, который выводит st = s, и определим GetBits(st), чтобы вычислить tt(st), проанализируем результат как str»σ с $|st^r| = n$, и выведем бит σ и обновленное состояние st^r. (Если мы используем этот потоковый шифр, чтобы генерировать p(n) выходные биты, начиная с начального числа s, то получим точно конечные p(n) битов tt^(s) в обратном порядке). Предшествующее доказательство показывает, что это дает псевдослучайный генератор.

Гибридные аргументы. Гибридный аргумент представляет собой базовый инструмент для доказательства неразличимости, когда базовый примитив (или несколько различных примитивов) применяется несколько раз. Отчасти произвольно, метод работает при определении ряда промежуточных «гибридных распределений», образуя связку между двумя «крайними распределениями», для которых нам требуется доказать неразличимость. (В приведенном выше доказательстве, эти крайние распределения соответствуют выходным данным tt^ и случайной строке). Чтобы применить метод доказательства, должны выполняться три условия. Во-первых, крайние распределения должны соответствовать исходным случаям, представляющим интерес. (В приведенном выше доказательстве, H_n^0 было равно распределению, индуцированному tt^, тогда как было равномерным распределением). Во-вторых, должно быть возможно транслировать возможность различения последовательных гибридных распределений в нарушение некоего основного предположения. (Выше мы показали, что отличие H_n^1 от H_n^{i+1} было эквивалентно отличию выхода tt от случайного). И наконец, количество гибридных распределений должно быть полиномом. См. также теорему 7.32.

Построение псевдослучайных функций

Теперь покажем, как построить псевдослучайную функцию из любого (удвоенной длины) псевдослучайного генератора. Напомним, что псевдослучайная функция является эффективно вычисляемой, функцией F с ключом, которая не отличима от истинной случайной функции, в том смысле, который описан в разделе 3.5.1. Для простоты, ограничим наш интерес случаем, где F сохраняет длину, это означает, что для $k \in \{0, 1\}^n$ функция F_k отображает n-битные входы на n-битные выходы. Псевдослучайная (сохраняющая длину) функция может рассматриваться, с неформальной точки зрения, как псевдослучайный генератор с коэффициентом расширения n • 2n; с учетом такого псевдослучайного генератора

tt мы могли бы определить, что Fk (i) (для $0 \le i < 2^n$) будет i-м n-битным блоком tt(k). Причина того, что это не работает, состоит в том, что F должна быть эффективно вычисляемой; существует экспоненциальное множество блоков и нам требуется способ, вычислить i-й блок, не вычисляя все другие блоки.

Мы будем делать это, вычисляя «блоки» выходных данных, идя вниз по двоичному дереву. Сначала проиллюстрируем конструкцию, показав псевдослучайную функцию, использующую 2-битные входы. Пусть tt будет псевдослучайным генератором с коэффициентом расширения 2^n. Если использовать tt как в доказательстве теоремы 7.20, то можно получить псевдослучайный генератор tt^ с коэффициентом расширения 4n , который использует три инициирования tt. (Мы создаем n дополнительных псевдослучайных битов каждый раз, когда применяется tt). Если определить, что F^r (i) (где $0 \le i < 4$ и i кодируется как 2-битная двоичная строка) будет i-м блоком tt^(k), то вычисление F^r (3) потребует вычисления всех tt^ и, следовательно, трех инициализаций tt. Теперь покажем, как построить псевдослучайную функцию F, используя только две инициализации tt на любом входе.

Пусть tt_0 и tt_1 будут функциями, обозначающими первую и вторую половину выхода tt; т.е., $tt(k) = tt_0(k) \mathbin{«} tt_1(k)$, где $|tt_0(k)| = |tt_1(k)| = |k|$. Определим F следующим образом:

$F_k(00) = tt_0(tt0(k))$ $\quad$ $F_k(10) = tt_0(tt1(k))$

$F_k(01) = tt_1(tt0(k))$ $\quad$ $F_k(11) = tt_1(tt1(k))$.

Мы утверждаем, что четыре строки, приведенные выше, являются псевдослучайными, даже если их рассматривать вместе. (Этого достаточно, чтобы доказать, что F –псевдослучайная). Интуитивно понятно, что это происходит потому, $tt_0(k) \| tt_1(k) = tt(k)$ – псевдослучайно и, следовательно, неотличимо от 2n-битной строки $k_0 \| k_1$. Но тогда

$tt_0(tt_0(k)) \| tt_1(tt_0(k)) \| tt_0(tt_1(k)) \| tt_1(tt_1(k))$ неотличимо от

$tt_0(k0) \| tt_1(k_0) \| tt0(k_1) \| tt_1(k_1) = tt(k_0) \| tt(k_1)$.

Поскольку tt – это псевдослучайный генератор, приведенное выше неотличимо от равномерной 4n-битной строки. Формальное доказательство использует гибридный аргумент.

Обобщая эту идею, мы можем получить псевдослучайную функцию на n-битных входах, определив

$F_k(x) = tt_{xn}(\bullet\bullet\bullet tt_{x1}(k)\bullet\bullet\bullet)$,

где x = x1 • • • xn; см. конструкцию 7.21. Интуитивное понимание, почему данная функция псевдослучайная то же, что и ранее, но формальное доказательство осложняется тем фактом, что теперь имеется экспоненциальное множество рассматриваемых входов.

КОНСТРУКЦИЯ 7.21

Пусть tt будет псевдослучайным генератором с коэффициентом расширения $A(n) = 2^n$, определим tt_0, tt_1 как в тексте. Для $k \in \{0,1\}^n$, определим функцию $Fk : \{0,1\}^n \to \{0,1\}^n$ как

$$F_{k(x1x2 \bullet\bullet\bullet xn)} = tt_{xn}\ (\bullet\bullet\bullet (tt_{x2}\ (tt_{x1}\ (k))) \bullet\bullet\bullet)\ .$$

Псевдослучайная функция из псевдослучайного генератора.

Полезно просмотреть эту конструкцию, как определяющую, для каждого ключа $k \in \{0, 1\}^n$, полное двоичное дерево высоты n, в котором каждый узел содержит n-битное значение.

(См. рис. 7.2, в котором n = 3.) Корень имеет значение k, и для каждого внутреннего узла со значением kr его левый дочерний элемент имеет значение tt0(kr), а его правый дочерний элемент имеет значение $tt_1(k^r)$. Результат Fk (x) для $x = x_1 \bullet\bullet\bullet x^n$ определяется, тогда как значение на концевом узле, достигнутом при движении по дереву в соответствии с битами x, где $x_i = 0$ означает «иди налево» и $x_i = 1$ означает «иди направо». (Эта функция определена только для входных данных длиной n, и поэтому только значения на концевых узлах всегда являются выходными данными). Размер дерева – степень n. Тем не менее, чтобы вычислить Fk (x) не нужно строить или сохранять все дерево; необходимы только n оценок tt.

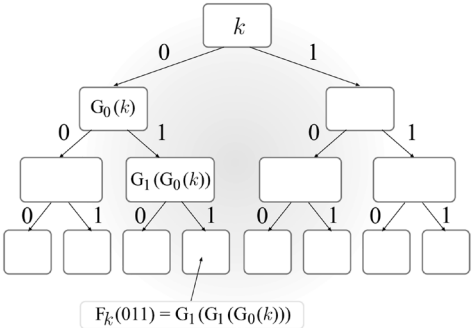

РИС. 7.2: Построение псевдослучайной функции.

ТЕОРЕМА 7.22 *Если tt является псевдослучайным генератором с коэффициентом расширения $A(n) = 2^n$, то конструкция 7.21 является псевдослучайной функцией.*

ДОКАЗАТЕЛЬСТВО Сначала мы покажем, что для любого полинома t невозможно отличить t(n) равномерные 2n-битные строки от t(n) псевдослучайных строк; т.е., для любого полинома t и любого ppt алгоритма A, следующее пренебрежимо мало:

$$\left| \Pr\left[A\left(r_1 \| \cdots \| r_{t(n)}\right) = 1\right] - \Pr\left[A\left(G(s_1) \| \cdots \| G(s_{t(n)})\right) = 1\right] \right|,$$

где первая вероятность – более равномерный выбор $r_1, \ldots,$ rt(n)$\in \{0, 1\}^{2n}$, вто-

рая вероятность – более равномерный выбор $s_1, \ldots, st(n) \in \{0,1\}^n$.

Доказательство происходит с помощью гибридного аргумента. Зафиксируем полином t и ppt алгоритм A, и рассмотрим следующий алгоритм A^r:

Отличительный признак A^r:

A^r задан в качестве входной строки $w \in \{0,1\}^{2n}$.

1. Выберем равномерное $j \in \{1, \ldots, t(n)\}$.

2. Выберем равномерные, независимые значения $r_1, \ldots, r_{j-1} \in \{0,1\}$ и $s_{j+1}, \ldots, st(n) \in \{0,1\}_n$.

3. Output A .$r_1 \| \bullet\bullet\bullet \| r_{j-1} | w | tt(s_{j+1}) | \bullet\bullet\bullet | tt(st(n))$..

Для любых n и $0 \le i \le t(n)$, пусть tt_i обозначает распределение строк длиной $2n \bullet t(n)$, в которых первые i «блоков» длиной 2^n являются равномерными и остальные $t(n) - i$ блоки являются псевдослучайными. Обратите внимание, что ttt(n) соответствует распределению, в котором все блоки t(n) равномерные, тогда как tt0 соответствует распределению, в котором все t(n) блоки псевдослучайные. То есть,

$$\left| \Pr_{y \leftarrow G_n^{t(n)}}[A(y)=1] - \Pr_{y \leftarrow G_n^0}[A(y)=1] \right| \qquad (7.11)$$
$$= \left| \Pr\left[A\left(r_1 \| \cdots \| r_{t(n)}\right)=1\right] - \Pr\left[A\left(G(s_1) \| \cdots \| G(s_{t(n)})\right)=1\right] \right|$$

Пусть Ar выбирает $j = j^*$. Если вход w – равномерная 2n-битная строка, то A запускается на входе в соответствии с $tt_n^{j^*}$. Если, с другой стороны, $w = tt(s)$ для равномерного s, то A действует на входе, распределенном в соответствии с tt_{j-1}. Это означает, что

$$\Pr_{r \leftarrow \{0,1\}^{2n}}[A'(r)=1] = \frac{1}{t(n)} \cdot \sum_{j=1}^{t(n)} \Pr_{y \leftarrow G_n^j}[A(y)=1]$$

$$\Pr_{s \leftarrow \{0,1\}^n}[A'(G(s))=1] = \frac{1}{t(n)} \cdot \sum_{j=0}^{t(n)-1} \Pr_{y \leftarrow G_n^j}[A(y)=1].$$

Следовательно,

$$\left| \Pr_{r \leftarrow \{0,1\}^{2n}}[A'(r)=1] - \Pr_{s \leftarrow \{0,1\}^n}[A'(G(s))=1] \right| \qquad (7.12)$$
$$= \frac{1}{t(n)} \cdot \left| \Pr_{y \leftarrow G_n^{t(n)}}[A(y)=1] - \Pr_{y \leftarrow G_n^0}[A(y)=1] \right|.$$

Поскольку tt представляет собой псевдослучайный генератор и Ar работает в полиномиальном времени, мы знаем, что левая сторона выражения (7.12) должна быть пренебрежимо малой; поскольку t(n) – полином, это означает, что

левая часть выражения (7.11) тоже пренебрежимо мала.

Обращаясь к сути доказательства, покажем, что F, как и в конструкции 7.21, является псевдослучайной функцией. Пусть D произвольный ppt отличительный признак, который задан 1n в качестве входных данных. Покажем, что D не может найти различий между случаем, когда задан прогнозирующий доступ к функции, которая равна Fk для равномерного k, или функцией, выбранной равномерно из $Func_n$. (См. раздел 3.5.1.) Для этого мы используем другой гибридный аргумент. Определим последовательность распределений, по значениям на концевых элементах всего двоичного дерева, высоты n. При соединении каждого концевого элемента со строкой длиной n, как и в конструкции 7.21, можно так же увидеть их в виде распределений по функциям, отображающим n-битные входы на n-битные выходы. Для любого n и $0 \le i \le n$, пусть H_i – следующее распределение по значениям на концевых элементах двоичного дерева высотой n: сначала выберем значения для узлов на уровне i , независимо и равномерно из $\{0, 1\}^n$. Тогда для каждого узла на уровне i или ниже со значением k, его левому дочернему элементу задано значение $tt_0(k)$, а его правому дочернему элементу задано значение $tt_1(k)$. Обратите внимание, что Hn соответствует распределению, в котором все значения на концевых элементах выбираются равномерно и независимо, и, таким образом, соответствуют выбору равномерной функции из Funcn, тогда как H0 соответствует выбору равномерного ключа k в конструкции 7.21, поскольку в этом случае только корень (на уровне 0) выбирается равномерно. То есть,

$$\left| \Pr_{k \leftarrow \{0,1\}^n}[D^{F_k(\cdot)}(1^n) = 1] - \Pr_{f \leftarrow \mathsf{Func}_n}[D^{f(\cdot)}(1^n) = 1] \right|$$
$$= \left| \Pr_{f \leftarrow H_n^0}[D^{f(\cdot)}(1^n) = 1] - \Pr_{f \leftarrow H_n^n}[D^{f(\cdot)}(1^n) = 1] \right|. \qquad (7.13)$$

Завершая доказательство, покажем, что выражение (7.13) пренебрежимо мало.

Пусть t = t(n) – полиномиальная верхняя граница числа запросов D, сделанных по их прогнозу на входе 1n. Определим отличительный признак A, который пытается отличить t(n) равномерные 2n-битные строки от t(n) псевдослучайных строк, следующим образом:

Отличительный признак A:

A задан в качестве входной 2n • t(n)-битной строки $w_1 \| \bullet \bullet \bullet \| w_t(n)$.

1. Выберем равномерное $j \in \{0, \ldots, n-1\}$. В дальнейшем A (неявно) поддерживает двоичное дерево высоты n, с n-битными значениями (подмножества) внутренних узлов на высоте j + 1 и ниже.

2. Запустим $D(1^n)$. Когда D делает запрос прогноза $x = x_1 \bullet \bullet \bullet x_n$, обратите внимание на префикс x1 • • • xj . Есть два случая:

• Если D до сих пор никогда не делал запроса с этим префиксом, то используйте $x_1 \bullet\bullet\bullet x_j$, чтобы достичь узла v на j-м уровне дерева. Возьмем следующую неиспользуемую 2n-битную строку w и зададим значение левого дочернего элемента узла v левой половине w , а значение правого дочернего элемента v правой половине w.

• Если D ранее делал запрос с префиксом $x_1 \bullet\bullet\bullet x_j$, то узлу $x_1 \bullet\bullet\bullet x_{j+1}$ уже было присвоено значение.

Используя значение у узла $x_1 \bullet\bullet\bullet x_{j+1}$, вычислим значение у концевого элемента в соответствии с $x_1 \bullet\bullet\bullet x_n$, как в конструкции 7.21, и возвратим это значение D.

3. Когда выполнение D завершено, выведем бит, возвращенный D.

A действует в течение полиномиального времени. Очень важно, что в этом случае A не нужно сохранять все двоичное дерево экспоненциального размера. Вместо этого оно «заполняется» значениями не более 2t(n) узлов в дереве. Пусть A выбирает $j = j^*$. Заметим, что:

1. Если вход A является равномерной $2^n \bullet t(n)$-битной строкой, то ответы, которые она дает D распределены точно так же, как если бы D взаимодействовало с функцией, выбранной из распределения H_{j+1}. Это справедливо, поскольку значения узлов на уровне дерева $j* + 1$ равномерны и независимы.

2. Если вход A состоит из t(n) псевдослучайных строк, т.е. wi = tt(si) для равномерного начального числа si, то ответы, которые он дает D распределены точно так же, как если бы D взаимодействовало с функцией, выбранной из распределения H_j. Это справедливо, поскольку значения узлов на уровне j* дерева (а именно, s-значения) равномерны и независимы. (Эти s-значения неизвестны для A, но это не имеет никакого значения).

Действуя, как и раньше, можно показать, что

$$\left| \Pr_{k\leftarrow\{0,1\}^n}[D^{F_k(\cdot)}(1^n)=1] - \Pr_{f\leftarrow \mathsf{Func}_n}[D^{f(\cdot)}(1^n)=1] \right|$$
$$= \left| \Pr_{f\leftarrow H_n^0}[D^{f(\cdot)}(1^n)=1] - \Pr_{f\leftarrow H_n^n}[D^{f(\cdot)}(1^n)=1] \right|. \qquad (7.13)$$

Ранее мы показали, что выражение (7.14) должно быть пренебрежимо малым. Сказанное выше подразумевает, что выражение (7.13) также должно быть пренебрежимо малым.

Построение (строгой) псевдослучайной перестановки

Далее мы покажем, как псевдослучайные перестановки и строгие псевдослучайные перестановки могут быть построены из любой псевдослучайной функции. Напомним из раздела 3.5.1, что псевдослучайная перестановка – это псевдослучайная функция, которая также эффективно обратима, тогда как строгую псевдослучайную перестановку трудно отличить от случайной перестановки,

даже когда оппоненту предоставлен доступ к прогнозу, как для самой перестановки, так и для ее инверсии

Еще раз о сетях Фейстеля. Сеть Фейстеля, представленная в разделе 6.2.2, обеспечивает способ построения обратимой функции из произвольного множества функций. Сеть Фейстеля работает в серии раундов. Входные данные для i-го раунда, представляют собой строку длиной 2n, разделенную на две n-битных половины L_{i-1} и R_{i-1} («левая половина» и «правая половина», соответственно). Выход i-го раунда – 2n-битная строка (L_i, R_i), где

$$L_i := R_{i-1} \text{ и } R_i := L_{i-1} \oplus fi(R_{i-1})$$

для некоторой вычисляемой (но не обязательно обратимой) функции, fi, отображающей n-битные входы на n-битные выходы. Обозначим через $Feistel_{f1,...,fr}$ r-й раунд сети Фейстеля, используя функции $f_1, \ldots, f_r$. (То есть, $Feistel_{f1,...,fr}(L_0, R_0)$ выводит 2n-битную строку (L_r, R_r)). В разделе 6.2.2 мы видели, что $Feistel_{f1,...,fr}$ является эффективной обратимой перестановкой независимо от $\{f_i\}$.

Можно определить ключевую перестановку с использованием сети Фейстеля, в которой {fi} зависит от ключа. Например, пусть $F : \{0,1\}^n \times \{0,1\}^n \rightarrow \{0,1\}^n$ – псевдослучайная функция, тогда определим ключевую перестановку $F^{(1)}$ как

$$F_k^{(1)} \stackrel{def}{=} (x) = Feistel_{Fk}(x).$$

(Обратите внимание, F (1) имеет n-битный ключ и отображает 2n-битные входы на 2n-битные выходы). Является ли F (1) псевдослучайной? Небольшое размышление показывает, что это явно не так. Для любого ключа $k \in \{0,1\}^n$ первые n битов выхода $F_k^{(1)}$ (то есть, L_1) равны последним n битам входа (т.е., R_0), это то, что происходит только при пренебрежимо малой вероятности для случайной функции.

Попытаемся еще раз, определим $F : \{0,1\}^{2n} \times \{0,1\}^{2n} \rightarrow \{0,1\}^{2^n}$ следующим образом:

$$F_{k1k2}^{(2)}(x) \stackrel{def}{=} Feistel_{Fk1k2}(x). \qquad (7.15)$$

(Заметим, что k_1 и k_2 являются независимыми ключами). К сожалению, F (2) – не псевдослучайная в любом случае, как вас просили показать в упражнении 7.16.

С учетом этого, может быть несколько удивительным то, что трехраундовая сеть Фейстеля является псевдослучайной. Определим ключевую перестановку F (3), взяв ключ длиной 3n и отобразив 2n-битные входы на 2n-битные выходы, следующим образом:

$$F_{k1k2k3}^{(3)}(x) \stackrel{def}{=} Feistel_{Fk1k2k3}(x)$$

где еще раз, k1, k2 и k3 – независимые. Имеем:

ТЕОРЕМА 7.23*Если F является псевдослучайной функцией, то F (3) является псевдослучайной перестановкой.*

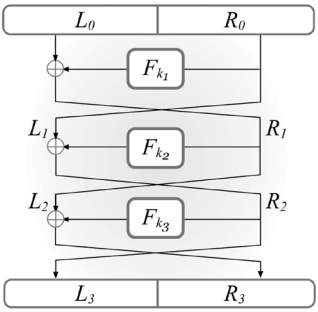

РИС. 7.3: Трехраундовая сеть Фейстеля, используемая для построения псевдослучайной перестановки из псевдослучайной функции.

ДОКАЗАТЕЛЬСТВО В стандартном способе мы можем заменить псевдослучайные функции, используемые при построении F (3), функциями, выбранными равномерно наугад. Псевдослучайность F предполагает, что функция имеет лишь пренебрежимо малое влияние на выход любого вероятностного полиномиально-временного отличительного признака, взаимодействующего с $F^{(3)}$ в качестве прогноза. Оставляем детали в качестве упражнения.

Пусть D – вероятностный полиномиально-временной отличительный признак. В оставшейся части доказательства покажем, что следующее пренебрежимо мало:

$$\left|\Pr[D^{\mathsf{Feistel}_{f_1,f_2,f_3}(\cdot)}(1^n)=1]-\Pr[D^{\pi(\cdot)}(1^n)=1]\right|$$

где первая вероятность берется по равномерному и независимому выбору f_1, f_2, f_3 из Funcn, а вторая вероятность берется по равномерному выбору π из Perm_{2n}. Зафиксируем некоторое значение для параметра безопасности n, и пусть q = q(n) обозначает полином верхней границы количества запросов прогноза, сделанных D. Без потери общности, предположим, что D никогда не делает один и тот же запрос прогноза дважды. Сосредоточив внимание на взаимодействии D с $\mathrm{Feistel}_{f1,f2,f3}(\bullet)$, допустим, что (L^i_0, R^i_0) обозначает i-й запрос, который D делает по своему прогнозу, и допустим, что (L^i_3, R^i_3), (L^i_1, R^i_1), и (L^i_2, R^i_2) обозначает промежуточные значения после раундов 1, 2 и 3, соответственно, являющиеся результатом этого запроса. (См. рис. 7.3). Обратите внимание, что D выбирает (L^i_0, R^i_0) и видит результат L^i_3, R^i_3), но прямо не видит (L^i_1, R^i_1) или (L^i_2, R^i_2).

Мы говорим, что имеется коллизия при R_1, если $R^i_1 = R^j_1$ для некоторого отдельного i, j. Сначала докажем, что коллизия при R1 возникает лишь с пренебрежимо малой вероятностью. Рассмотрим любое фиксированное, отдельное i, j. Если $Ri_1 = Rj_1$, то $L^i_0 \neq L^j_0$, но тогда

$$R_1^i = L_0^i \oplus f_1(R_0^i) \neq L_0^j \oplus f_1(R_0^j) = R_1^j.$$

Если $R_0^i = R_0^i$, то $f_1(R_0^i)$ и $f_1(R_0^i)$ являются равномерными и независимыми, таким образом,

$$\Pr\left[L_0^i \oplus f_1(R_0^i) = L_0^j \oplus f_1(R_0^j)\right] = \Pr\left[f_1(R_0^j) = L_0^i \oplus f_1(R_0^i) \oplus L_0^j\right] = 2^{-n}.$$

Учет границы объединения по всем отдельным i, j показывает, что вероятность коллизии при R_1 не более $q^2/2^n$.

Скажем, существует коллизия при R_2, если $R_2^i = R_2^i$ для некоторого отдельного i, j. Докажем, что обусловленная отсутствием коллизии при R_1, вероятность коллизии при R_2 пренебрежимо мала. Проведем анализ, как указано выше: рассмотрим любое фиксированное i, j и отметим, что если отсутствует коллизия при R_1, то $R_1 i = R_1 i$. Таким образом, $f_2(R_1 i)$ и $f_2(R_1 i)$ являются равномерными и независимыми, и поэтому

$$\Pr\left[L_1^i \oplus f_2(R_1^i) = L_1^j \oplus f_2(R_1^j) \mid \text{no collision at } R_1\right] = 2^{-n}$$

(Обратите внимание, что f_2 независимо от f_1, что облегчает указанное выше вычисление). Учет границы объединения по всем отдельным i, j дает

$\Pr[\text{коллизии при } R_2 \mid \text{отсутствие коллизии при } R_1] \le q^2/2^n$.

Обратите внимание, что $L_3^i = R_2^i = L_1^i \oplus f_2(R_1^i)$; таким образом, обусловлено отсутствие какой-либо коллизии при R_1, все значения $L_1, \ldots, Lq$ независимы и равномерно распределены в $\{0, 1\}^n$. Если дополнительно выдвинуть условие для события отсутствия коллизии при R_2, то значения $L1, \ldots, Lq$ равномерно распределены среди всех последовательностей q различных значений в $\{0, 1\}^n$. Аналогично, $Ri = Li \oplus f3(Ri)$; таким образом, обусловлено отсутствие коллизии при R_2, все значения $R_1, \ldots, R_q$ равномерно распределены в $\{0, 1\}^n$, а также независимы друг от друга $L_1, \ldots, L_q$. Подведем итог: при запросе $F^{(3)}$ (с равномерными раундовыми функциями) на серии q различных входов, кроме как с пренебрежимо малой вероятностью, выходные значения $(L_1, R_1), \ldots, (Lq, Rq)$ распределены так, что $\{L^i\}$ являются равномерными и независимыми, но различными, n-битными значениями, и $\{R^i\}$ являются равномерными и независимыми n-битными значениями. В противоположность этому, при запросе случайной перестановки на серии q различных входов, выходные значения $(L1, R1), \ldots, (Lq, Rq)$ равномерные и независимые, но различные 2n-битные значения. Лучшая отличительная атака для D, состоит в том, чтобы догадаться, что он взаимодействует со случайной перестановкой, если $Li = Lj$ для некоторого отдельного i, j. Но данное событие возникает с пренебрежимо малой вероятностью, в этом случае. Это можно превратить в формальное доказательство.

$F^{(3)}$ не является строгой псевдослучайной перестановкой, что будет предложено вам продемонстрировать в упражнении 7.17. К счастью, добавление четвертого раунда дает в результате, строгую псевдослучайную перестановку. Детали приведены в виде конструкции 7.24.

ТЕОРЕМА 7.25 *Если F – псевдослучайная функция, то конструкция 7.24 является строгой псевдослучайной перестановкой, которая отображает 2n-битные входы на 2n-битные выходы (и использует 4n-битный ключ).*

КОНСТРУКЦИЯ 7.24

Пусть F будет сохраняющей длину функцией с ключом. Определим ключевую перестановку $F^{(4)}$ следующим образом:

• Входы: Ключ $k=(k_1, k_2, k_3, k_4)$ при $|k_i|=n$, и вход $x \in \{0,1\}^{2n}$ анализируется как (L_0, R_0) при $|L_0|=|R_0|=n$.

• Вычисление:

1. Вычислим $L_1 := R_0$ и $R_1 := L_0 \oplus Fk_1(R_0)$.
2. Вычислим $L_2 := R_1$ и $R_2 := L1 \oplus Fk_2(R_1)$.
3. Вычислим $L_3 := R_2$ и $R_3 := L_2 \oplus Fk_3(R_2)$.
4. Вычислим $L_4 := R_3$ и $R_4 := L_3 \oplus Fk_4(R_3)$.
5. Выведем (L_4, R_4).

Строгая псевдослучайная перестановка из любой псевдослучайной функции.

Допущения для криптографии с закрытым ключом.

Мы показали, что: (1) если существуют односторонние перестановки, то существуют псевдослучайные генераторы; (2) если существуют псевдослучайные генераторы, то существуют псевдослучайные функции; и (3) если существуют псевдослучайные функции, то существуют (строгие) псевдослучайные перестановки. Несмотря на то, что мы это здесь не доказали, возможно построить псевдослучайные генераторы из односторонних функций. Таким образом, имеем следующую фундаментальную теорему:

ТЕОРЕМА 7.26 *Если существуют односторонние функции, то существуют также псевдослучайные генераторы, псевдослучайные функции и строгие псевдослучайные перестановки.*

Все схемы с закрытым ключом, которые мы изучали в главах 3 и 4, могут быть построены из псевдослучайных генераторов/функций. Таким образом, мы имеем:

ТЕОРЕМА 7.27 *Если существует односторонняя функция, то также существуют схемы шифрования с закрытым ключом, безопасные с точки зрения атаки на основе подобранного шифрованного текста и аутентификационные коды безопасных сообщений.*

То есть, односторонних функций достаточно для всей криптографии с закрытым ключом.

Здесь мы покажем, что односторонние функции также необходимы.

Псевдослучайность предполагает односторонние функции. Сначала покажем, что псевдослучайные генераторы подразумевают существование односто-

ронних функций:

ПРЕДПОЛОЖЕНИЕ 7.28 *Если существует псевдослучайный генератор, то существует также и односторонняя функция.*

ДОКАЗАТЕЛЬСТВО Пусть tt – псевдослучайный генератор с коэффициентом расширения $A(n) = 2^n$. (Из теоремы 7.20, мы знаем, что существование псевдослучайного генератора предполагает существование псевдослучайного генератора с коэффициентом расширения). Покажем, что сам tt является односторонним. Эффективная вычисляемость проста (поскольку tt можно вычислить в течение полиномиального времени). Покажем, что способность инвертировать tt может быть преобразована в способность отличать выход tt от равномерного. Интуитивно понятно, что это справедливо, поскольку способность инвертировать tt подразумевает способность найти начальное число, используемое генератором.

Пусть A – произвольный вероятностный полиномиально-временной алгоритм. Покажем, что $\Pr[\mathrm{Invert}_{A,G}(n) = 1]$ пренебрежимо мало (см. определение 7.1). Чтобы убедиться в этом, рассмотрим следующий ppt отличительный признак D: на входе строка $w \in \{0,1\}^{2n}$, запустим A(w), чтобы получить выход s. Если tt(s) = w, то выход 1; иначе, выход 0.

Проанализируем поведение D. Сначала рассмотрим вероятность того, что D выводит 1, когда его входная строка w является равномерной. Поскольку существует не более 2^n значений в диапазоне tt (а именно, значения $\{tt(s)\}_{s\in\{0,1\}^n}$), то вероятность того, что w находится в диапазоне tt составляет не более $2^n/2^{2n} = 2^{-n}$. Если w не находится в диапазоне tt, то для A невозможно вычислить инверсию w и, таким образом, невозможно для D вывести 1. Приходим к выводу, что

$$\Pr_{w\leftarrow\{0,1\}^{2n}}[D(w) = 1] \leq 2^{-n}.$$

С другой стороны, если w = tt(s) для начального числа $s \in \{0,1\}^n$ выбраны равномерно и случайно, то по определению, A вычисляет правильную инверсию (и, таким образом, D выводит 1) с вероятностью точно равной $\Pr[\mathrm{Invert}_{A,G}(n) = 1]$. Таким образом,

$$\left|\Pr_{w\leftarrow\{0,1\}^{2n}}[D(w) = 1] - \Pr_{s\leftarrow\{0,1\}^{n}}[D(G(s)) = 1]\right| \geq \Pr[\mathsf{Invert}_{\mathcal{A},G}(n) = 1] - 2^{-n}.$$

Поскольку tt представляет собой псевдослучайный генератор, то указанное выше должно быть пренебрежимо малым. Поскольку 2^{-n} пренебрежимо мало, то это предполагает, что $\Pr[\mathrm{Invert}_{A,G}(n) = 1]$ также пренебрежимо мало и, таким образом, tt является односторонним.

Нетривиальное шифрование с закрытым ключом предполагает односторонние функции. Предположение 7.28 не подразумевает, что односторонние функции необходимы для построения схем шифрования с закрытым клю-

чом, поскольку их можно построить, не используя псевдослучайный генератор. Кроме того, можно строить совершенно секретные схемы шифрования (см. главу 2), до тех пор, пока открытый текст не длиннее, чем ключ. Таким образом, доказательство того, что безопасное шифрование с закрытым ключом подразумевает использование односторонних функций, требует большей тщательности.

ПРЕДПОЛОЖЕНИЕ 7.29 *Если существует схема шифрования с закрытым ключом, безопасная с точки зрения расширенной проверки приложения (EAV), которая шифрует сообщения в два раза длиннее ее ключа, то существует и односторонняя функция.*

ДОКАЗАТЕЛЬСТВО Пусть $\Pi = (Enc, Dec)$ – схема шифрования с закрытым ключом, которая имеет неотличимые шифровки в присутствии перехватчика и шифрует сообщения длиной 2n, ключ имеет длину n. (Для простоты, предположим, что ключ выбран равномерно). Допустим, что когда используется n-битный ключ, Enc использует не более A(n) случайных битов. Обозначим шифрование сообщения m, использующее ключ k и степень случайности r by $Enc_k(m; r)$.

Определим следующую функцию f :

$$f(k, m, r) \stackrel{def}{=} Enc(m; r) \| m,$$

где $|k| = n$, $|m| = 2n$ и $|r| = A(n)$. Покажем, что f является односторонней функцией. Очевидно, что ее можно эффективно вычислить; покажем, что ее трудно инвертировать. Допуская, что A – произвольный ppt алгоритм, покажем, что $Pr[Invert_{A,f}(n) = 1]$ пренебрежимо мало (см. определение 7.1).

Рассмотрим следующего вероятностного полиномиально-временного оппонента Ar, атакующего схему шифрования с закрытым ключом Π (т.е., в эксперименте $PrivK^{eav}_{\mathcal{A},\Pi}$

Оппонент $A^r(1^n)$

1. Выберем равномерное $m_0, m_1 \leftarrow \{0, 1\}^{2n}$ и выведем его. Примем в ответ на вызов зашифрованный текст c.

2. Запустим $A(c \| m_0)$, чтобы получить (k^r, m^r, r^r). Если $f(k^r, m^r, r^r) = c \| m_0$, выведем 0; в противном случае выведем 1.

Теперь проанализируем поведение A^r. Когда c является шифровкой m0, то c»m0 распределено равномерно как f (k, m0, r) равномерных k, m0 и r. Поэтому, A выводит действительную инверсию c»m0 (и, следовательно, Ar выводит 0) с вероятностью точно равной $Pr[Invert_{A,f}(n) = 1]$.

С другой стороны, когда c является шифровкой m1, то c не зависит от m0. Для любого фиксированного значения вызванного шифрованного текста c, существует не более 2^n возможных сообщений (по одному для каждого возможного ключа), которому c может соответствовать. Поскольку m0 является

равномерной 2n-битной строкой, что означает, что вероятность того, что существует некий ключ k, для которого Dec_k (c) = m0 составляет не более $2^n/22^n = 2^{-n}$. Это дает верхнюю границу по вероятности, с которой A может вывести действительную инверсию c|| m_0 при f, и, следовательно верхнюю границу по вероятности, с которой A^r выводит 0 в этом случае.

Сопоставив все указанное выше, имеем:

$$\begin{aligned}
&\Pr\left[\mathsf{PrivK}^{\mathsf{eav}}_{\Pi,\mathcal{A}'}(n) = 1\right] \\
&= \frac{1}{2} \cdot \Pr\left[\mathcal{A}' \text{ outputs } 0 \mid b = 0\right] + \frac{1}{2} \cdot \Pr\left[\mathcal{A}' \text{ outputs } 1 \mid b = 1\right] \\
&\geq \frac{1}{2} \cdot \Pr[\mathsf{Invert}_{\mathcal{A},f}(n) = 1] + \frac{1}{2} \cdot \left(1 - 2^{-n}\right) \\
&= \frac{1}{2} + \frac{1}{2} \cdot \left(\Pr[\mathsf{Invert}_{\mathcal{A},f}(n) = 1] - 2^{-n}\right) .
\end{aligned}$$

Безопасность П означает, что Pr[$\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\Pi}$ = 1]. ≤ 1/2+negl(n) для некоторой пренебрежимо малой функции negl. Это, в свою очередь, подразумевает, что $Pr[Invert_{A,f}(n) = 1]$ пренебрежимо мало; это завершает доказательство того, что f является односторонней.

Коды идентификации сообщений предполагают односторонние функции. Верно также, что коды аутентификации сообщений, удовлетворяющие определению 4.2, предполагают существование односторонних функций. Как и в случае шифрования с закрытым ключом, доказательство этого факта несколько утонченное, поскольку нестандартные коды аутентификации сообщений существуют, если существует теоретическая граница количества сообщений, которые будут проверены на подлинность. (См. раздел 4.6). Таким образом, доказательство опирается на тот факт, что оппонент видит теги аутентификации на произвольном (полиноме) количестве сообщений. Это доказательство несколько запутано, поэтому мы не приводим его здесь.

Обсуждение. Мы пришли к выводу, что существование односторонних функций необходимо и достаточно для всей (нетривиальной) криптографии с закрытым ключом. Другими словами, односторонние функции – минимальное допущение, когда речь идет о криптографии с закрытым ключом. Интересно, что, по-видимому, это не относится к хэш-функциям и шифрованию с открытым ключом, где, как известно, односторонние функции необходимы, но не известно (предполагается), что достаточны.

Вычислительная неразличимость

Понятие вычислительной неразличимости занимает центральное место в теории криптографии и она лежит в основе большей части того, что мы видели в главе 3 и в данной главе. С неформальной точки зрения, два распределения вероятности являются вычислительно неразличимыми, если никакой эффективный алгоритм не может разделить их (или различить их). Более подробно, рас-

смотрим два распределения X и Y по строкам некоторой длины A; то есть, каждый X и Y присваивает некую вероятность каждой строке $\{0,1\}^A$. Говоря, что некоторый алгоритм D не может отличить эти два распределения, мы имеем в виду, что D не может сообщить, является ли данная строка отобранной в соответствии с распределением X или является ли данная строка отобранной в соответствии с распределением Y . Иными словами, если представить D, выводящий «0», когда считается, что его входные данные были отобраны в соответствии с X и выводящий «1», если предполагается, что его входные данные были отобраны в соответствии с Y , тогда вероятность того, что D выводит «1» должна быть приблизительно той же, независимо от того, откуда D обеспечен образцом – из X или из Y . Другими словами, мы хотим, чтобы

$$\left| \Pr_{s \leftarrow X}[D(s) = 1] - \Pr_{s \leftarrow Y}[D(s) = 1] \right|$$

было малым.

Это должно напоминать способ, которым мы определили псевдослучайные генераторы и, действительно, скоро мы формально заново определим понятие псевдослучайного генератора, используя данную терминологию.

Формальное определение вычислительной неразличимости относится к вероятностным ансамблям, которые представляют собой бесконечные последовательности распределений вероятности. (Этот формализм необходим для осмысленного асимптотического подхода). Хотя это понятие можно обобщить, мы рассматриваем вероятностные ансамбли, в которых основные распределения индексированы натуральными числами. Если для каждого натурального числа n существует распределение X_n, то $X = \{X_n\}_{n \in N}$ является вероятностным ансамблем. Часто бывает так, что $X_n = Y_{t(n)}$ для некоторой функции t, в этом случае мы записываем $\{Y_{t(n)}\}_{n \in N}$ вместо $\{X_n\}_{n \in N}$.

Нас интересуют только «эффективно отбираемые» вероятностные ансамбли. Ансамбль $X = \{X_n\}_{n \in N}$ является эффективно отбираемым, если существует вероятностный полиномиально-временной алгоритм S , такой что случайные переменные $S(1^n)$ и X_n распределены идентично. То есть, алгоритм S является эффективным способом выбора X . Теперь можно формально определить, что означает для двух ансамблей, быть вычислительно неразличимыми.

ОПРЕДЕЛЕНИЕ 7.30 *Два вероятностных ансамбля* $X = \{X_n\}_{n \in N}$ и $Y = \{Y_n\}_{n \in N}$ *являются вычислительно неразличимыми, что обозначается как* $X \equiv Y$, *если для каждого вероятностного полиномиально-временного отличительного признака D существует пренебрежимо малая функция negl , такая что:*

$$\left| \Pr_{x \leftarrow X_n}[D(1^n, x) = 1] - \Pr_{y \leftarrow Y_n}[D(1^n, y) = 1] \right| \leq \mathsf{negl}(n).$$. Pr

В определении D задан одинарный вход 1^n, поэтому он может работать в течение полиномиального времени n. Это важно, когда выходы X_n и Y_n могут иметь длину менее n. В качестве условного обозначения в вероятностных выражениях, мы будем иногда записывать X, в качестве заполнителя для случайного образца из расределения X. То есть, будем записывать $\Pr[D(1^n, X_n) = 1]$ вместо $\Pr_{x \leftarrow X}[D(1^n, x) = 1]$.

Отношение вычислительной неразличимости, транзитивно: if $X \equiv Y$ и $Y \equiv Z$, то $X \equiv Z$.

Псевдослучайность и псевдослучайные генераторы. Псевдослучайность – это просто частный случай, вычислительной неразличимости. Пусть для любого целого A, U_A обозначает равномерное распределение по $\{0, 1\}^A$. Мы можем определить псевдослучайный генератор следующим образом:

ОПРЕДЕЛЕНИЕ 7.31 Пусть A(•) будет полиномом, а tt – (детерминированным) полиномиально-временным алгоритмом, где для всех s справедливо, что |tt(s)| = A(|s|). Мы говорим, что tt – псевдослучайный генератор, если выполняются два следующих условия:

1. **(Расширение:)** *Для каждого n справедливо, что A(n) > n.*

2. **(Псевдослучайность:)** *Ансамбль $\{tt(U_n)\}_{n\in N}$ вычислительно неотличим от ансамбля $\{UA(n)\}_{n\in N}$.*

Многие из других определений и допущений в этой книге, также можно рассматривать как частные случаи или варианты вычислительной неразличимости.

Несколько примеров. Важная теорема о вычислительной неразличимости заключается в том, что полиномиально многие образцы (эффективно отбираемые) вычислительно неразличимых ансамблей, также являются вычислительно неразличимыми.

ТЕОРЕМА 7.32 Пусть X и Y будут эффективно отбираемыми вероятностными ансамблями, которые являются вычислительно неразличимыми.. Тогда для каждого полинома p, ансамбль $\overline{\mathcal{X}} = \{(X_n^{(1)}, \ldots, X_n^{(p(n))})\}_{n\in\mathbb{N}}$ является вычислительно неотличимым от ансамбля $\overline{\mathcal{Y}} = \{(Y_n^{(1)}, \ldots, Y_n^{(p(n))})\}_{n\in\mathbb{N}}$

Например, пусть tt будет псевдослучайным генератором с коэффициентом расширения 2n, в этом случае ансамбли $\{tt(U_n)\}_{n\in N}$ и $\{U_{2n}\}_{n\in N}$ будут вычислительно неразличимыми. В доказательстве теоремы 7.22 мы показали, что для любого полинома t, ансамбли

$$\{(\underbrace{G(U_n), \ldots, G(U_n)}_{t(n)})\}_{n\in\mathbb{N}} \quad \text{and} \quad \{(\underbrace{U_{2n}, \ldots, U_{2n}}_{t(n)})\}_{n\in\mathbb{N}}$$

также являются вычислительно неразличимыми. Теорема 7.32 доказывается с помощью гибридного аргумента, точно тем же способом.

Ссылки и дополнительная литература

Понятие односторонней функции впервые было предложено Диффи (Diffie) и Хеллменом (Hellman) [58] и позднее формализовано Яо (Yao) [179]. Трудные предикаты были введены Блумом (Blum) и Микали (Micali) [37], а факт существования трудного предиката для каждой односторонней функции был доказан Голдрейхом (Goldreich) и Левином (Levin) [79]. Первая конструкция псевдослучайного генератора (при специфическом теоретико-числовом предположении трудности) была дана Блумом (Blum) и Микали (Micali) [37]. Построение псевдослучайного генератора из любой односторонней перестановки, было приведено Яо (Yao) [179], а тот результат, что псевдослучайные генераторы могут быть построены из односторонней функции, был показан Хостадом и др. (H°astad et al.) [85]. Псевдослучайные функции были определены и построены Голдрейхом (Goldreich), Голдвассером (Goldwasser) и Микали (Micali) [78], а их расширение до (строгой) псевдослучайной перестановки показали Люби (Luby) и Ракофф (Rackoff) [116]. Тот факт, что односторонние функции являются необходимым допущением для большей части криптографии с закрытым ключом, было показано в [93]. Доказательсто теоремы 7.29 взято из работы [72].

На наше представление сильно повлияла книга Голдрейха (Goldreich) [75], которую мы настоятельно рекомендуем тем, кто заинтересован в более подробном изучении темы данной главы.

Упражнения

Докажите, что если существует односторонняя функция, то существует односторонняя функция f, такая что $f(0n) = 0n$ для каждого n. Обратите внимание, что для бесконечного множества значений y, легко вычислить $f^{-1}(y)$. Почему это не противоречит односторонности?

Докажите, что если f является односторонней функцией, то функция g, определенная с помощью $g(x_1, x_2) \stackrel{def}{=} (f(x_1), x_2)$, где $|x_1| = |x_2|$, также односторонняя функция.

Заметим, что g показывает половину своих входных данных, но, тем не менее, является односторонней.

Докажите, что если существует односторонняя функция, то существует односторонняя функция, сохраняющая длину.

> **Подсказка:** Пусть f будет односторонней функцией и p(•) полиномом, так что $|f(x)| \leq p(|x|)$. (Обоснуйте существование такого p). Определите $f'(x) \stackrel{def}{=} f(x)\|10^{p(|x|)-|f(x)|}$. Модифицируйте далее f', чтобы получить функцию, сохраняющую длину, которая остается односторонней.

Пусть (Gen, H) – устойчивая к коллизиям функция хэширования, где H отображает строки длины 2^n в строки длины n. Докажите, что семейство функций

(Gen, Samp, H) является односторонним (см. определение 7.3), где Samp – тривиальный алгоритм, который отбирает равномерную строку длиной 2n.

Подсказка: Выбор равномерного $x \in \{0,1\}^{2n}$ и поиск инверсии $y = Hs(x)$ не гарантирует коллизии. Однако это дает коллизию в большую часть времени.

Пусть F –(сохраняющая длину) псевдослучайная перестановка.

(a) Покажите, что функция $f(x, y) = F_x(y)$ не является односторонней.

(b) Покажите, что функция $f(y) = F_{0^n}(y)$ (где $n = |y|$) не является односторонней.

(c) Докажите, что функция $f(x) = F_x(0^n)$ (где $n = |x|$) является односторонней.

Пусть f – сохраняющая длину односторонняя функция, а hc – трудный предикат f . Определите tt как $tt(x) = f(x)\|hc(x)$. tt обязательно представляет собой псевдослучайный генератор? Докажите свой ответ.

Докажите, что односторонние функции существуют тогда и только тогда, когда существуют семейства односторонних функций. Обсудите, почему доказательство ваше, не переносится на случай односторонних перестановок.

Пусть f – односторонняя функция. Является ли $g(x) \stackrel{\text{def}}{=} f(f(x))$ обязательно односторонней функцией? А что можно сказать о $gr(x) \stackrel{\text{def}}{=} f(x)\|f(f(x))$? Докажите свои ответы.

Пусть $\Pi = (Gen, Samp, f)$ – семейство функций. Функция $hc: \{0,1\}^* \to \{0,1\}$ является трудным предикатом Π, если он является эффективно вычисляемым и если для каждого ppt алгоритма A существует пренебрежимо малая функция negl, такая что

$$\Pr_{I \leftarrow \mathsf{Gen}(1^n),\, x \leftarrow \mathsf{Samp}(I)} [\mathcal{A}(I, f_I(x)) = \mathsf{hc}(I, x)] \leq \frac{1}{2} + \mathsf{negl}(n).$$

Докажите версию теоремы Голдрейха-Левина для этого задания, а именно, если семейство односторонних функций Π (соотв., перестановок) существует, то тогда существует семейство односторонних функций Π^r (соотв., перестановок) и трудный предикат hc of Π^r.

Покажите построение псевдослучайного генератора из семейства односторонних перестановок. Можно использовать результат предыдущего упражнения.

Это упражнение предназначено для студентов, которые прошли курс теории сложности или другим образом ознакомились с NP полнотой.

(a) Покажите, что существование односторонних функций подразумевает $P \neq NP$.

(b) Допустим, что $P \neq NP$. Покажите, что существует функция f , такая что: (1) может быть вычислена в течение полиномиального времени, (2) ее трудно инвертировать в худшем случае (т.е., для всего вероятностного полиномиально-временного A, $\Pr_{x \leftarrow \{0,1\}^n}[f(A(f(x))) = f(x)] \neq 1$), но (3) не является односторонней.

Пусть x $\in \{0, 1\}^n$ и обозначим x = $x_1 \bullet \bullet \bullet x_n$. Докажите, что если существует односторонняя функция, то существует односторонняя функция f, такая что для каждого i существует алгоритм Ai, такой что

$$\Pr_{x \leftarrow \{0,1\}^n}[A_i(f(x)) = x_i] \geq \frac{1}{2} + \frac{1}{2n}$$

(Это упражднение показывает, что невозможно утверждать, что каждая односторонняя функция скрывает, по меньшей мере, один специфический бит входных данных).

Покажите, что если односторонняя функция f имеет трудный предикат, то f является односторонней.

Покажите, что если конструкция 7.21 изменяется естественным способом так, что Fk (x) определено для каждой непустой строки x, длиной не более n, то эта конструкция не длиннее псевдослучайной функции.

Докажите, что если существует псевдослучайная функция такая, что использование ключа длиной n, отображает n-битные входы одноразрядных выходов, то существует псевдослучайная функция, которая отображает n-битные входы n-битных выходов.

Подсказка: Используйте ключ длиной n2 и, используя гибридный аргумент, докажите, что ваша конструкция безопасна.

Докажите, что двухраундовая сеть Фейстеля, использующая псевдослучайные раундовые функции (как в выражении (7.15)), не является псевдослучайной.

Докажите, что трехраундовая сеть Фейстеля, использующая псевдослучайные раундовые функции (как в выражении (7.16)), не является строго псевдослучайной.

Подсказка: Это значительно труднее, чем в предыдущем упражнении. Используйте отличительный признак, который делает два запроса к перестановке, и один запрос к ее инверсии.

Рассмотрим ключевую перестановку F * , определенную с помощью

$F^*_k \stackrel{def}{=}$ (x) = $\text{Feistel}_{Fk, Fk, Fk}$ (x).

(Обратите внимание, что в каждом раунде используется тот же самый ключ). Покажите, что F* не является псевдослучайной.

Пусть tt – псевдослучайный генератор с коэффициентом расширения A(n) = n + 1. Докажите, что tt является односторонней функцией.

Пусть X , Y, Z – вероятностный ансамбль. Докажите, что если X ≡ Y и Y ≡ Z, то X ≡ Z.

Докажите теорему 7.32.

Пусть X = $\{X_n\}_{n\in N}$ и Y = $\{Y_n\}_{n\in N}$ – вычислительно неразличимые вероятностные ансамбли. Докажите, что для любого вероятностного полиномиально-временного алгоритма A, ансамбли $\{A(X_n\}_{n\in N}$ и $\{A(Y_n\}_{n\in N}$ вычислительно неразличимы.

Индекс общего обозначения

Общие обозначения:

• := называется детерминированным присвоением

• Если S – множество, то x ← S обозначает, что x выбирается равномерно из S

• Если A – случайный алгоритм, то y ← A(x) обозначает действующий A на входе x, с равномерной случайной лентой , и назначающий выход для y. Мы пишем y := A(x; r), чтобы обозначить действующий A на входе x , использующий случайную ленту r , и назначающий выход для y

• ∧ обозначает логическое умножение (оператор И (AND))

• V обозначает логическое отрицание (оператор ИЛИ (OR))

• ⊕ обозначает исключающее ИЛИ (оператор XOR); этот оператор может быть применен к отдельным битам или целым строкам (в последнем случае оператор XOR действует поразрядно)

• $\{0, 1\}^n$ это множество всех битовых строк длиной n

• $\{0, 1\}^{\leq n}$ то множество всех битовых строк длиной не более n

• $\{0, 1\}^*$ множество конечных битовых строк

• 0^n (соотв., 1n) обозначает строку, составленную из n нулей (соотв., n единиц)

• $\|x\|$ обозначает длину двоичного представления (положительного) целого числа x, записанного вместе с ведущим битом 1. Обратите внимание, что $\log x < \|x\| \leq \log x + 1$

• |x| обозначает длину двоичной строки x (у которой могут быть ведущие нули), или абсолютное значение действительного числа x

• O(•), Θ(•), Ω(•), ω(•) см. приложение A.2

• 0x обозначает, что эти числа представлены в шестнадцатеричном коде

• x||y однозначную конкатенацию строк x и y («однозначная» означает, что x and y могут быть восстановлены из x||y)

• Pr[X] обозначает вероятность события X

• log x обозначает логарифм x по основанию 2

Запись, специфическая для криптографии:

• n – параметр безопасности

• ppt означает в течение «вероятностного полиномиального времени»

• $A^{O(\bullet)}$ обозначает алгоритм A с доступом к прогнозу для O

• k, как правило, обозначает секретный ключ (как в шифровании с закрытым ключом и кодами аутентификации сообщений)

• (pk, sk) обозначает пару ключей открытый/закрытый (для шифрования с открытым ключом и цифровых подписей)

• negl обозначает пренебрежимо малую функцию; см. определение 3.4

• poly(n) обозначает произвольный полином

• polylog(n) обозначает poly(log(n))

• Funcn обозначает множество функций, отображающих n-битные строки для n-битных строк

• Permn обозначает множество взаимно однозначных соответствий на n-битных строках

• IV обозначает вектор инициализации (используется для режимов работы и устойчивых к коллизиям хэш-функций)

Алгоримы и процедуры:

• tt обозначает псевдослучайный генератор

• F обозначает функцию с ключом, т.е., как правило, псевдослучайную функцию или перестановку

• (Gen, Enc, Dec) обозначает процедуры генерации ключа, шифрования и дешифрования, соответственно, как для шифрования с открытым ключом, так и для шифрования с закрытым ключом Для случая шифрования с закрытым ключом, когда Gen не указан, то Gen(1n) выводит равномерное $k \in \{0, 1\}^n$

• (Gen, Mac, Vrfy) обозначает процедуры генерации ключа, генерации тега и верификации, соответственно, для аутентификационного кода сообщения. Когда Gen не указан, то Gen(1n) выводит равномерное $k \in 0, 1\}^n$

• (Gen, Sign, Vrfy) обозначает процедуры генерации ключа, генерации подписи и верификации, соответственно, для схемы цифровой подписи

• GenPrime обозначает ppt алгоритм, который на входе 1^n, выводит n-битное простое число, кроме как при пренебрежимо малой вероятности в n

• GenModulus обозначает ppt алгоритм, который на входе 1^n выводит (N, p, q), где N= pq и (кроме как, при пренебрежимо малой вероятности) p и q являются n-битными простыми числами

• GenRSA обозначает ppt алгоритм, который на входе 1n, выводит (кроме как, при пренебрежимо малой вероятности) модуль N , целое $e > 0$ с наибольшим общим делителем $(e, \varphi(N)) = 1$, и целое d, удовлетворяющее $ed = 1 \bmod \varphi(N)$

• G обозначает ppt алгоритм, который на входе 1n выводит (кроме как, при пренебрежимо малой вероятности) описание циклической группы G, группового порядка q (с $||q|| = n$), и генератор $g \in G$.

Теория чисел:

• Z обозначает множество целых чисел

• a | b означает, что a делит b

• af | b означает, что a не делит b

• gcd(a, b) обозначает наибольший общий делитель для a и b

• [a mod b] обозначает остаток a после деления на b. Обратите внимание, что $0 \leq [a \bmod b] < b$.

• $x_1 = x_2 = \bullet\bullet\bullet = xn \bmod N$ означает, что $x_1, \ldots, x_n$ все являются конгруэнтными по модулю N

Примечание: $x = y \bmod N$ означает, что x и y конгруэнтны по модулю N , тогда как $x = [y \bmod N]$ означает, что x равно остатку от y при делении на N .

• Z_N обозначает аддитивную группу целых чисел по модулю N , а также множество $\{0, \ldots, N-1\}$

• Z_N^* обозначает мультипликативную группу обратимых целых чисел по модулю N (т.е., числа, которые взаимно просты по отношению к N)

• $\varphi(N)$ обозначает размер Z_N^*

• G и H обозначают группы

• G_1 с G_2 означает, что группы G1 and G2 изоморфны. Если этот изоморфизм задан f и $f(x_1) = x_2$, то мы пишем $x_1 \leftrightarrow x_2$

• g является типичным генератором группы

• logg h обозначает дискретный логарифм h по основанию g

• (g) обозначает группу, сгенерированную с помощью g

• p и q обычно обозначают простые числа

• N обычно обозначает произведение двух простых чисел p и q одинаковой длины

• QRp является множеством квадратичных вычетов по модулю p

• QN Rp является множеством квадратичных невычетов по модулю p

• $J_p(x)$ является символом Якоби для x по модулю p

• J_N^{+1} является множеством элементов с символом Якоби +1 по модулю N

• J_N^{-1} – это множество элементов с символом Якоби −1 по модулю N

• QN R_N^{+1} – это множество квадратичных невычетов по модулю N , имеющих символ Якоби +1

Приложение А

Математическое обоснование

Тождества и неравенства

Перечислим некоторые стандартные тождества и неравенства, которые используются в различных местах по всему тексту.

ТЕОРЕМА А.1 *(Теорема биномиального разложения) Пусть x, y – действительные целые числа n – положительное целое. Тогда*

$$(x+y)^n = \sum_{i=0}^{n} \binom{n}{i} x^i\, y^{n-i}.$$

ПРЕДПОЛОЖЕНИЕ А.2 *Для всех $x \geq 1$ справедливо, что $(1 - 1/x)^x \leq e^{-1}$.*

ПРЕДПОЛОЖЕНИЕ А.3 *Для всех x справедливо, что $1 - x \leq e^{-x}$.*

ПРЕДПОЛОЖЕНИЕ А.4 Для всех x при $0 \leq x \leq 1$ справедливо, что

$$e^{-x} \leq 1 - \left(1 - \frac{1}{e}\right) \cdot x \leq 1 - \frac{x}{2}.$$

Асимптотическая запись

Мы используем стандартную запись для выражения асимптотического поведения функций.

ОПРЕДЕЛЕНИЕ А.5 *Пусть f (n), g(n) будут функциями из неотрицательных целых чисел , для неотрицательных действительных чисел. Тогда:*

- *$f(n) = O(g(n))$ означает, что существуют положительные целые числа c и nr, такие что для всех $n > nr$ справедливо, что $f(n) \leq c \cdot g(n)$.*
- *$f(n) = \Omega(g(n))$ означает, что существуют положительные целые числа c и nr, такие, что для всех $n > nr$ справедливо, что $f(n) \geq c \cdot g(n)$.*
- *$f(n) = \Theta(g(n))$ означает, что существуют положительные целые числа c_1, c_2 и n^r такие, что для всех $n > n^r$ справедливо, что $c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$.*
- *$f(n) = o(g(n))$ означает, что $\lim n \to \infty\ g(n) = 0$.*
- *$f(n) = \omega(g(n))$ означает, что $\lim n \to \infty\ g(n) = \infty$.*

Пример А.6

Пусть $f(n) = n^4 + 3n + 500$. Тогда:

- $f(n) = O(n^4)$.

• $f(n) = O(n^5)$. На самом деле, $f(n) = o(n^5)$.

• $f(n) = \Omega(n3 \log n)$. Фактически, $f(n) = \omega(n3 \log n)$.

• $f(n) = \Theta(n^4)$. ♦

Основы теории вероятности

Мы предполагаем, что читатель знаком с основами теории вероятности на уровне, который рассматривается в типичном университетском курсе по дискретной математике. Здесь мы просто напомним читателю о некоторых обозначениях и основных фактах.

If E – это событие, то $\bar{E}$ обозначает дополнение этого события; т.е., $\bar{E}$ – это событие, которое E не происходит. По определению, $Pr[E] = 1 - Pr[\bar{E}]$. Если E_1 и E_2 являются событиями, то $E_1 \wedge E_2$ обозначает их конъюнкцию; т.е., $E_1 \wedge E2$ является таким событием, что происходит как E_1, так и E_2 . По определению, $Pr[E_1 \vee E_2] \leq Pr[E_1]$. События E_1 и E_2 называются независимыми, если $Pr[E_1 \vee E_2] = Pr[E_1] \cdot Pr[E_2]$.

Если E_1 и E_2 – события, то $E_1 \vee E_2$ обозначает логическое сложение (дизъюнкцию) E_1 и E_2; то есть, $E_1 \vee E_2$ – такое событие, при котором происходит или E_1 или E_2. Из определения следует, что $Pr[E_1 \vee E_2] \geq Pr[E_1]$. Граница объединения часто является очень полезной верхней границей этой величины.

ПРЕДПОЛОЖЕНИЕ A7 (граница объединения)

Повторное применение границы объединения для любых событий $E_1, \ldots, Ek$ дает

$$\Pr\left[\bigvee_{i=1}^{k} E_i\right] \leq \sum_{i=1}^{k} \Pr[E_i].$$

Условная вероятность E1 с учетом E2, обозначенная Pr[E1 | E2], определяется как

$$\Pr[E_1 \mid E_2] \stackrel{\text{def}}{=} \frac{\Pr[E_1 \wedge E_2]}{\Pr[E_2]}$$

пока $Pr[E_2] \neq 0$. (Если $Pr[E_2] = 0$, то $Pr[E_1 \mid E_2]$ не определено). Это представляет вероятность того, что событие E1 происходит с учетом того, что событие E2 произошло. Непосредственно из определения следует, что

$Pr[E_1 \wedge E_2] = Pr[E_1 \mid E_2] \cdot Pr[E_2]$;

равенство справедливо, если $Pr[E_2] = 0$ до тех пор, пока мы интерпретируем умножение на 0 с правой стороны обычным образом.

Теперь можно легко вывести теорему Байеса.

ТЕОРЕМА A.8 (Теорема Байеса) Если $Pr[E_2] \neq 0$, то

$$\Pr[E_1 \mid E_2] = \frac{\Pr[E_2 \mid E_1] \cdot \Pr[E_1]}{\Pr[E_2]}$$

ДОКАЗАТЕЛЬСТВО Это следует из того, что

$$\Pr[E_1 \mid E_2] = \frac{\Pr[E_1 \wedge E_2]}{\Pr[E_2]} = \frac{\Pr[E_2 \wedge E_1]}{\Pr[E_2]} = \frac{\Pr[E_2 \mid E_1] \cdot \Pr[E_1]}{\Pr[E_2]}$$

Пусть n – события, такие что $\Pr[E_1 \vee \bullet\bullet\bullet \vee E_n] = 1$ и $\Pr[E_i \wedge E_j] = 0$ для всех i *f*= j. То есть {Ei} – разбиение пространства всех возможных событий, так что с вероятностью 1 одно из событий E_i, точно происходит. Тогда для любого F

$$\Pr[F] = \sum_{i=1}^{n} \Pr[F \wedge E_i].$$

Особый случай, когда n = 2 и $E_2 = \bar{E}_1$, давая $\Pr[F]=\Pr[F\wedge E_1]+\Pr[F\wedge \bar{E}_1]$ $= \Pr[F \mid E_1] \bullet \Pr[E_1] + \Pr[F \mid \bar{E}_1] \bullet \Pr[\bar{E}_1]$.

Принимая $F=E_1 \vee E_2$, получим более строгую версию границы объединения: $\Pr[E_1 \vee E_2] = \Pr[E_1 \vee E_2 \mid E_1] \bullet \Pr[E_1] + \Pr[E_1 \vee E_2 \mid \bar{E}_1] \bullet \Pr[\bar{E}_1]$ $\leq \Pr[E_1] + \Pr[E_2 \mid \bar{E}_1]$.

Распространив это на события $E_1, \ldots, En$, получим

ПРЕДПОЛОЖЕНИЕ А.9

$$\Pr\left[\bigvee_{i=1}^{k} E_i\right] \leq \Pr[E_1] + \sum_{i=2}^{k} \Pr[E_i \mid \bar{E}_1 \wedge \cdots \wedge \bar{E}_{i-1}].$$

***Полезные вероятностные границы**

Рассмотрим некоторые термины и положения вероятностных границ, которые являются стандартными, но могут не встречаться в базовом курсе дискретной математики. Приведенный здесь материал используется только в разделе 7.3.

Случайная (дискретная, действительная) величина X является переменной, значение которой присваивается вероятностно из некоторого конечного множества S действительных чисел. X – неотрицательная величина, если не принимает отрицательных значений; это 0/1-случайная переменная, если S = {0, 1}. 0/1-случайные переменные $X_1, \ldots, X_k$ являются независимыми, если для всех $b_1, \ldots, b_k$ справедливо, что $\Pr[X_1 = b_1 \wedge \bullet\bullet\bullet \wedge X_k = b_k] = Qk\ \Pr[X_i = b_i]$. Обозначим как Exp[X] математическое ожидание случайной величины X; если X принимает в множестве S, то $\mathrm{Exp}[X] \stackrel{\mathrm{def}}{=} \sum_{s\in S} s \bullet \Pr[X = s]$. Одним из наиболее важных фактов является то, что математическое ожидание линейно ; для случайных величин $X_1, \ldots, X_k$ (с произвольными зависимостями) имеем $\mathrm{Exp}[\sum_i X_i] = \sum_i \mathrm{Exp}[X_i]$. Если X_1, X_2 – независимые, то $\mathrm{Exp}[X_i \bullet X_j] = \mathrm{Exp}[X_i] \bullet \mathrm{Exp}[X_j]$.

Неравенство Маркова полезно, когда о X известно мало.

ПРЕДПОЛОЖЕНИЕ А.10 (неравенство Маркова) *Пусть X – неотрицательная случайная величина и v > 0. Тогда Pr[X ≥ v] ≤ Exp[X]/v.*

ДОКАЗАТЕЛЬСТВО Допустим, что X принимает значения в множестве S. Имеем

$$\begin{aligned}\mathsf{Exp}[X] &= \sum_{s\in S} s\cdot \Pr[X=s]\\ &\geq \sum_{x\in S,\, x<v} \Pr[X=s]\cdot 0 + \sum_{x\in S,\, x\geq v} v\cdot \Pr[X=s]\\ &= v\cdot \Pr[X\geq v].\end{aligned}$$

Дисперсия X, обозначенная как Var[X], определяет, насколько X отклоняется от своего математического ожидания. Имеем $\mathrm{Var}[X] \stackrel{\text{def}}{=} \mathrm{Exp}[(X - \mathrm{Exp}[X])^2] = \mathrm{Exp}[X^2] - \mathrm{Exp}[X]^2$, и можно легко показать, что $\mathrm{Var}[aX + b] = a^2\mathrm{Var}[X]$. Для 0/1-случайной величины X_i, имеем $\mathrm{Var}[X_i] \leq 1/4$, поскольку в этом случае $\mathrm{Exp}[X_i] = \mathrm{Exp}[X^2]$ и, таким образом, $\mathrm{Var}[X_i] = \mathrm{Exp}[X_i](1-\mathrm{Exp}[X_i])$, которая становится максимальной при $\mathrm{Exp}[X_i] = 1$.

ПРЕДПОЛОЖЕНИЕ A.11 (неравенство Чебышева) Пусть X – случайная величина и $\delta > 0$. Тогда:

$$\Pr[|X - \mathsf{Exp}[X]| \geq \delta] \leq \frac{\mathsf{Var}[X]}{\delta^2}$$

ДОКАЗАТЕЛЬСТВО Определим неотрицательную случайную величину Y $\stackrel{\text{def}}{=}$ $(X - \mathrm{Exp}[X])^2$ и затем применим неравенство Маркова. Таким образом,

$$\begin{aligned}\Pr[|X - \mathsf{Exp}[X]| \geq \delta] &= \Pr[(X - \mathsf{Exp}[X])^2 \geq \delta^2]\\ &\leq \frac{\mathsf{Exp}[(X - \mathrm{Exp}[X])^2]}{\delta^2} = \frac{\mathsf{Var}[X]}{\delta^2}.\end{aligned}$$

0/1-случайные величины $X_1, \ldots, X_m$ are pairwise independent, если для каждого i *f*= j и каждого bi, bj∈{0, 1} справедливо, что

$\Pr[X_i = bi \wedge X_j = b_j] = \Pr[X_i = b_i] \bullet \Pr[X_j = b_j]$.

Если $X_1, \ldots, X_m$ попарно независимы, то $\mathsf{Var}[\sum_{i=1}^m X_i] = \sum_{i=1}^m \mathsf{Var}[X_i]$ (Это следует, поскольку $\mathrm{Exp}[X_i \bullet X_j] = \mathrm{Exp}[X_i] \bullet \mathrm{Exp}[X_j]$ когда i *f*= j, используя попарную независимость). Важное следствие неравенства Чебышева приведено ниже.

СЛЕДСТВИЕ A.12 Пусть $X_1, \ldots, X_m$ – попарно независимые случайные величины с одним и тем же математическим ожиданием μ и дисперсией σ^2. Тогда для каждого $\delta > 0$,

$$\Pr\left[\left|\frac{\sum_{i=1}^m X_i}{m} - \mu\right| \geq \delta\right] \leq \frac{\sigma^2}{\delta^2 m}.$$

ДОКАЗАТЕЛЬСТВО При линейности математического ожидания $\mathsf{Exp}[\sum_{i=1}^m X_i/m]$ неравенство Чебышева для случайной величины. Применяя $\mathsf{Exp}[\sum_{i=1}^m X_i/m]$, имеем

$$\Pr\left[\left|\frac{\sum_{i=1}^{m} X_i}{m} - \mu\right| \geq \delta\right] \leq \frac{\mathsf{Var}\left[\frac{1}{m} \cdot \sum_{i=1}^{m} X_i\right]}{\delta^2}.$$

Из использования попарной независимости следует, что

$$\mathsf{Var}\left[\frac{1}{m} \cdot \sum_{i=1}^{m} X_i\right] = \frac{1}{m^2} \sum_{i=1}^{m} \mathsf{Var}[X_i] = \frac{1}{m^2} \sum_{i=1}^{m} \sigma^2 = \frac{\sigma^2}{m}.$$

Неравенство получено комбинацией двух приведенных выше выражений.

Пусть 0/1-случайные величины $X_1, \ldots, X_m$, каждая из которых дает оценку некоторого фиксированного (неизвестного) бита b. То есть, $\Pr[X_i = b] \geq 1/2 + \varepsilon$ для всех i, где $\varepsilon > 0$.

Можно оценить b, рассмотрев значение X_1; эта оценка будет корректной с вероятностью $\Pr[X_1 = b]$. Более точную оценку можно получить, рассмотрев значения $X_1, \ldots, X_m$ и взяв значение, которое появляется большую часть времени. Проанализируем, насколько хорошо это выполняется, когда $X_1, \ldots, X_m$ являются попарно независимыми.

ПРЕДПОЛОЖЕНИЕ А.13 *Зафиксируем $\varepsilon > 0$ и $b \in \{0, 1\}$, и пусть $\{X_i\}$ будут попарно независимыми, 0/1-случайными величинами, для которых $Pr[X_i = b] \geq 1 + \varepsilon$ для всех i. Рассмотрим процесс, в котором m значений $X_1, \ldots, X_m$ записывают и X задают значение, которое появляется точно в большую часть времени. Тогда*

$$\Pr[X \neq b] \leq \frac{1}{4 \cdot \varepsilon^2 \cdot m}$$

ДОКАЗАТЕЛЬСТВО Допустим b = 1; в случае симметрии, это не ведет к потере общности. Тогда $\mathrm{Exp}[X_i] = 1/2 + \varepsilon$. Пусть X обозначает точное большинство $\{X_i\}$ как в предположении, и отметим, что X f= 1 если и только если $\sum_{i=1}^{m} X_i \leq m/2$. Таким образом,

$$\begin{aligned}
\Pr[X \neq 1] &= \Pr\left[\sum_{i=1}^{m} X_i \leq m/2\right] \\
&= \Pr\left[\frac{\sum_{i=1}^{m} X_i}{m} - \frac{1}{2} \leq 0\right] \\
&= \Pr\left[\frac{\sum_{i=1}^{m} X_i}{m} - \left(\frac{1}{2} + \varepsilon\right) \leq -\varepsilon\right] \\
&\leq \Pr\left[\left|\frac{\sum_{i=1}^{m} X_i}{m} - \left(\frac{1}{2} + \varepsilon\right)\right| \geq \varepsilon\right]
\end{aligned}$$

Поскольку $\mathrm{Var}[X_i] \leq 1/4$ для всех i, применяя предыдущий вывод, покажем, что $\Pr[X \neq 1] \leq \frac{1}{4\varepsilon^2 m}$ как и утверждалось.

Более точная граница получается, если $\{X_i\}$ независимы:

ПРЕДПОЛОЖЕНИЕ А.14 (граница Чернова) *Зафиксируем $\varepsilon > 0$ и $b \in \{0, 1\}$, и пусть $\{X_i\}$ будут независимыми 0/1-случайными величинами $Pr[X_i = b] = 1/2 + \varepsilon$ для всех i. Вероятность того, что значение большинства из них b составляет не более $e^{-\varepsilon m/2}$.*

Проблема «парадокса дней рождения»

Если выбрать q элементов $y_1, \ldots, y_q$ равномерно из множества размера N, то какова вероятность того, что существуют различные i, j при $y_i = y_j$? Мы называем указанное событие коллизией и обозначаем вероятность этого события как coll(q, N). Эта проблема относится к, так называемому, парадоксу дней рождения, когда задают вопрос о том, какого размера группа людей нужна, чтобы с вероятностью 1/2 у некой пары людей в группе совпадал день рождения. Чтобы увидеть это отношение, пусть y_i обозначает день рождения i-го человека в группе. Если в группе есть q человек, то мы имеем q значений $y1, \ldots, yq$ выбранных равномерно из $\{1, \ldots, 365\}$, делая упрощающее предположение, что дни рождения равномерно и независимо распределены среди 365 дней не високосного года. Кроме того, совпадающие дни рождения соответствуют коллизии, т.е., отличию i, j при $y_i = y_j$. Таким образом, нужное решение для проблемы дней рождения задается минимальным (целым) значением q, для которого $coll(q, 365) \geq 1/2$. (Ответ может удивить—достаточно q = 23 человек!)

В этом разделе мы докажем нижние и верхние границы coll(q, N). Взятые вместе и обобщенные на высоком уровне, они показывают, что если $q < \sqrt{N}$, то вероятность коллизии составляет $\Theta(q^2/N)$; иначе, для $q = \Theta(\sqrt{N})$ вероятность коллизии является константой.

Верхнюю границу для вероятности коллизии легко получить.

ЛЕММА А.15 *Для положительного целого N и, некоторого q, элементы y1, . . . , yq выбраны равномерно и независимо на случайной основе из множества размера N. Тогда вероятность того, что существуют различные i, j при yi = yj не превышает $q^2/2N$. То есть,*

$$\mathrm{coll}(q, N) \leq \frac{q^2}{2N}.$$

ДОКАЗАТЕЛЬСТВО Доказательство – это простое применение границы объединения (предположение А.7). Напомним, что коллизия означает, что существует различные i, j при $y_i = y_j$. Пусть Coll обозначает событие коллизии а $Coll_{i,j}$ обозначает событие, что $yi = yj$. Из этого прямо следует, что $Pr[Coll_{i,j}] = /N$ для любых различных i, j. Кроме того, $Coll = W_f\, Coll_{i,j}$ и, таким образом, повторное применение границы объединения подразумевает, что

$$\Pr[\mathsf{Coll}] = \Pr\left[\bigvee_{i\neq j} \mathsf{Coll}_{i,j}\right]$$
$$\leq \sum_{i\neq j} \Pr[\mathsf{Coll}_{i,j}] = \binom{q}{2} \cdot \frac{1}{N} \leq \frac{q^2}{2N}.$$

ЛЕММА А.16 *Зафиксируем положительное целов N и допустим, что q ≤√2N элементов y1, . . . , yq выбираются случайно равномерно и независимо, из множества, размера N. Тогда вероятность того, что существуют различные i, j при yi = yj составляет не менее* $\frac{q(q-1)}{4N}$.

$$\mathsf{coll}(q, N) \geq 1 - e^{-q(q-1)/2N} \geq \frac{q(q-1)}{4N}.$$

ДОКАЗАТЕЛЬСТВО Напомним, что коллизия означает, что существуют различные i, j при yi = yj. Пусть Coll обозначает это событие. Пусть NoColli будет событием, которое не является коллизией между y1, . . . , yi; то есть, yj *f*= yk для всех j < k ≤ i. Тогда NoCollq = tChoell является событием, когда никакой коллизии вообще нет.

Если NoCollq происходит, то NoColli также должно происходить для всех i ≤ q. Таким образом,

Pr[NoCollq]=Pr[NoColl1]•Pr[NoColl2|NoColl1]• • •Pr[NoCollq| NoCollq−1].

Теперь, Pr[NoColl1] = 1, поскольку y1 не может иметь коллизии с собой. Кроме того, если событие NoColli происходит, то {y1, . . . , yi} содержит i различные значения; таким образом, вероятность того, что yi+1 имеет коллизию с одним из этих значений равна i и, следовательно, вероятность того, что y_{i+1} не имеет коллизии с каким-либо из этих значений, равна1 −i/N. Это означает i Pr[NoColl_{i+1} | NoColli] = 1 − N , и, таким образом,

$$\Pr[\mathsf{NoColl}_q] = \prod_{i=1}^{q-1} \left(1 - \frac{i}{N}\right)$$

Поскольку i/N < 1 для всех i мы имеем 1 − i/N ≤ $e^{-i/N}$ (согласно неравенству А.3) и, таким образом,

$$\Pr[\mathsf{NoColl}_q] \leq \prod_{i=1}^{q-1} e^{-i/N} = e^{-\sum_{i=1}^{q-1}(i/N)} = e^{-q(q-1)/2N}$$

Мы приходим к выводу, что

$$\Pr[\mathsf{Coll}] = 1 - \Pr[\mathsf{NoColl}_q] \geq 1 - e^{-q(q-1)/2N} \geq \frac{q(q-1)}{4N}$$

используя неравенство А.4 на последнем шаге (обратите внимание, что q(q − 1)/2N < 1).

***Конечные поля**

Мы используем конечные поля в книге только эпизодически, но включили их определение и некоторые основные факты для полноты картины. Более подробную информацию можно найти в любом учебнике по абстрактной алгебре.

ОПРЕДЕЛЕНИЕ А.17 *(Конечное) поле является (конечным) множеством F вместе с двумя двоичными операциями +, • , для которых справедливо следующее:*

• F является коммутативной группой по отношению к операции '+.' Пусть 0 обозначает единичный элемент этой группы.

• F \ {0} является коммутативной группой по отношению к операции '•.' Пусть 1 обозначает единичный элемент этой группы.

Как обычно, мы часто пишем ab вместо a • b.

• (Дистрибутивность:) *Для всех a, b, c ∈ F имеем a • (b + c) = ab + ac.*

Аддитивная инверсия a ∈F, обозначаемая как −a, является уникальным элементом, удовлетворяющим a +(−a) = 0; мы пишем b − a вместо b +(−a). Мультипликативная инверсия a ∈ F \ {0}, обозначаемая как a^{-1}, является уникальным элементом, удовлетворяющим $aa^{-1} = 1$; мы часто пишем b/a вместо ba^{-1}.

Пример А.18

Из результатов раздела 8.1.4 следует, что для любого простого числа p множество {0, . . . , p − 1} является конечным полем, относительно сложения и умножения по модулю p. Мы обозначаем это поле как F_p. ♦

Для конечных полей имеется богатая теория. Для наших целей нам нужно лишь несколько основных фактов. Порядок F – это количество элементов в F (в предположении, что оно конечное). Напомним также, что q является степенью простого числа, если $q = p^r$ для некоторого простого числа p и целого $r \geq 1$.

ТЕОРЕМА А.19 *Если F является конечным полем, то порядок F представляет собой степень простого числа. С другой стороны, для каждой степени простого числа q существует конечное поле порядка q, которое , к тому же единственное такое поле (до переименования элементов).*

Для $q = p^r$ с простым числом p, пусть Fq обозначает (единственное) поле порядка q. Назовем p характеристикой Fq . Предыдущая теорема говорит нам о том, что характеристикой любого конечного поля является простое число.

Как и в случае групп, если n – положительное целое число и a ∈ F , то

$$n \cdot a \stackrel{\text{def}}{=} \underbrace{a + \cdots + a}_{n \text{ times}} \quad \text{and} \quad a^n \stackrel{\text{def}}{=} \underbrace{a \cdots a}_{n \text{ times}}.$$

Запись расширена для $n \leq 0$ естественным способом.

ТЕОРЕМА А.20 *Пусть F_q – конечное поле характеристики p. Тогда для всех $a \in F_q$ имеем $p \cdot a = 0$.*

Пусть $q = p^r$ с простым числом p. Для r = 1, мы видели в примере А.18, что Fq =Fp можно взять в виде множества $\{0, \ldots, p-1\}$ при сложении и умножении по модулю p. Однако мы предупреждаем, что r > 1 множество $\{0, \ldots, q-1\}$ не является полем при сложении и умножении по модулю q. Например, если взять $q = 3^2 = 9$, то элемент 3 не имеет мультипликативной инверсии по модулю 9. Конечные поля характеристики p можно представить, используя полиномы по Fp.

Приведем пример, чтобы продемонстрировать разновидность конструкции, не обсуждая вопрос, почему конструкция работает, и не описывая общий случай. Построим поле F_4, работая с полиномами по F_2. Зафиксируем полином $r(x) = x^2 + x + 1$ и заметим, что r(x) не имеет корней по F_2, поскольку r(0) = r(1) = 1 (напомним, что мы работаем в F_2, а это значит, что все операции выполняются по модулю 2). Таким же способом, каким мы можем ввести мнимое число i, которое будет корнем $x^2 + 1$ по полю действительных чисел, мы можем ввести значение ω, которое будет корнем r(x) по F_2; то есть, $\omega^2 = -\omega - 1$. Затем определим, что F_4 будет множеством всех полиномов степени-1 от ω по F_2; то есть, $F_4 = \{0, 1, \omega, \omega + 1\}$. Сложение в F_4 будет только регулярным сложением полинома, при этом следует помнить о том, что операции с коэффициентами выполняются в F2 (то есть, по модулю 2). Умножение в F_4 будет полиномиальным умножением (опять же, при операциях с коэффициентами, выполняемыми по модулю 2), следующих за подстановкой $\omega^2 = -\omega - 1$; это также гарантирует, что результат лежит в F_4. Так, например, $\omega + (\omega + 1) = 2\omega + 1 = 1$

и $(\omega + 1) \cdot (\omega + 1) = \omega^2 + 2\omega + 1 = (-\omega - 1) + 1 = -\omega = \omega$.

Несмотря на неочевидность, можно проверить, что это является полем; единственное трудное для проверки условие состоит в том, каждый ненулевой элемент имеет мультипликативную инверсию.

Только нам нужен единственный другой результат.

ТЕОРЕМА А.21 *Пусть Fq будет конечным полем порядка q. Тогда коммуникативная группа Fq \ {0} по отношению к '•' является циклической группой порядка q − 1.*

Приложение B

Основная алгоритмическая теория чисел

Для криптографических конструкций, приведенных в этой книге, чтобы быть эффективными (т.е., действовать в течение времени, являющимся полиномами длин их входов), необходимо чтобы эти конструкции использовали эффективные (т.е. полиномиально-временные) алгоритмы для выполнения основных теоретико-числовых операций. Хотя в некоторых случаях существуют «тривиальные» алгоритмы, которые будут работать, все же целесообразно внимательно рассмотреть их эффективность, поскольку для криптографических приложений нередко используются целые числа длиной тысячи битов. В других случаях получение какого-либо полиномиально-временного алгоритма требует немного сообразительности, а анализ их эффективности может опираться на нетривиальные теоретико-групповые результаты.

В приложении B.1 мы опишем основные алгоритмы для целочисленной арифметики. Здесь мы рассмотрим знакомые алгоритмы для сложения, вычитания и т.д., а также алгоритм Эвклида для вычисления наибольших общих делителей. Мы также обсудим расширенный алгоритм Эвклида, предполагая, что читатель ознакомился с материалом в разделе 8.1.1.

В приложении B.2 мы покажем различные алгоритмы для арифметических операций с абсолютными значениями чисел. В дополнение к краткому обсуждению основных операций по модулю (т.е. модульной редукции, сложения, умножения и инверсии), мы опишем умножение Монтгомери, которое может существенно упростить (и ускорить) реализацию операций модульной арифметики. Затем мы обсудим алгоритмы для проблем, которые менее распространены за пределами области криптографии: возведение в степень по модулю N (а также в произвольных группах) и выбор равномерного элемента из Z_N or Z_N^* (или в произвольной группе). Этот раздел предполагает знакомство с основами теории групп, рассмотренных в разделе 8.1.

Указанный выше материал, неявно используется во второй половине книги, несмотря на то, что абсолютной необходимости читать этот материал, чтобы следовать книге, нет. (В частности, читатель, готовый принять результаты этого приложения без доказательства, может просто прочитать краткое изложение этих результатов в теоремах, изложенных ниже). Приложение B.3, в котором обсуждается поиск генераторов в циклических группах (когда факторизация группового порядка известна) и предполагает результаты раздела 8.3.1, содержит материал,

который вряд ли вообще используется; он включен для полноты и ссылок.

Поскольку наша цель состоит в том, чтобы установить, что определенные проблемы могут быть решены в течение полиномиального времени, мы сделали выбор в пользу простоты, а не эффективности алгоритмов и их описаний (до тех пор, пока алгоритмы работают в течение полиномиального времени). По этой причине мы вообще не интересуемся точным временем работы алгоритмов, мы представляем, что после установления они действительно работают в течение полиномиального времени. Читатель, который серьезно заинтересован в реализации этих алгоритмов, предупрежден, что необходимо искать другие источники для более эффективных альтернатив, а также различные методы для ускорения необходимых вычислений.

Результаты этого приложения обобщены теоремами, которые приведены далее. Мы везде предполагаем, что любое целое a, представленное в качестве входных данных, записывается с использованием именно «a» битов; т.е. бит высокого порядка равный 1. В приложении B.1 мы покажем:

ТЕОРЕМА B.1 *(Операции с целыми числами) Даны целые числа a и b, можно выполнить следующие операции в течение полиномиального времени «a» и «b»:*

1. Вычисление суммы a + b и разности a − b;

2. Вычисление произведения ab;

3. Вычисление положительных целых чисел q и r < b, так что a = qb + r (т.е., вычисление деления с остатком);

4. Вычисление наибольшего целого делителя a и b, gcd(a, b);

5. Вычисление целых чисел X, Y при Xa + Y b = gcd(a, b).

Следующие результаты доказаны в приложении B.2:

ТЕОРЕМА B.2 (Модульные операции) *Даны целые числа N > 1, a и b, можно выполнить следующие операции в течение полиномиального времени в «a», «b», and «N «:*

1. Вычисление модульной редукции [a mod N];

2. Вычисление суммы [(a + b) mod N], разности [(a − b) mod N], и произведения [a^b mod N];

3. Определение, является ли a обратимой по модулю N ;

4. Вычисление мультипликативной инверсии [a−1 mod N], в предположении, что a обратимо по модулю N ;

5. Вычисление возведения в степень [ab mod N].

Нижеследующее обобщает теорему B.2(5) для произвольных групп:

ТЕОРЕМА B.3 (Групповое возведение в степень) *Пусть G – группа, за-*

писанная мультипликативно. Пусть g будет элементом этой группы и b неотрицательным целым числом. Тогда gb можно вычислить, используя poly(«b») групповые операции.

ТЕОРЕМА B.4 (Выбор равномерных элементов) *Существует вероятностный алгоритм со следующими свойствами: на входе N ,*

• Алгоритм работает в течение полиномиального времени в «N «;

• Алгоритм выводит fail (ошибка) с вероятностью пренебрежимо малой в «N «; а также

• Настроенный так, чтобы не выводить fail (ошибка), алгоритм выводит равномерно распределенный элемент Z_N .

Алгоритм с аналоговыми свойствами для Z_N также существует.*

Поскольку вероятность того, что любой алгоритм, упоминаемый в приведенной выше теореме, выводит fail (ошибка), пренебрежимо мала, мы игнорируем эту возможность (и вместо этого оставляем в неявном виде). В приложении B.2 также обсуждаются обобщения изложенного выше, для случая выбора равномерного элемента из какой-либо конечной группы (с учетом определенных требований к представлению элементов группы).

Доказательство нижеследующего приведено в приложении B.3:

ТЕОРЕМА B.5 *(Тестирование и нахождение генераторов) Пусть G будет циклической группой порядка q, предположим, что групповая операция и выбор однородных элементов группы могут быть выполнены в единицу времени.*

1. Существует алгоритм, который на входе q выполняет разложение на простые множители q, и элемент g∈ G, действует в течение времени poly(«q») и решает, является ли g генератором G.

2. Существует вероятностный алгоритм, который на входе q и разложении на простые множители q, действует в течение времени poly(«q») и выводит генератор G, кроме как с пренебрежимо малой вероятностью в «q». Настроенный на выходе генератор, равномерно распределен среди генераторов G.

Целочисленная арифметика

Основные операции

Мы начнем наше объяснение алгоритмической теории чисел с обсуждения сложения/вычитания, умножения и деления с остатком целых чисел. Небольшое размышление показывает, что все эти операции могут быть выполнены в течение полиномиального времени в длине входных данных, с использованием для этих задач стандартных алгоритмов «начальной школы». Например, сложение двух положительных целых чисел a и b, при a > b может быть выполнено в течение времени,

линейного в «a» пошагово, поочередно, по битам a ид b, начиная битами нижнего порядка и вычисляя соответствующий выходной бит и «текущий бит» на каждом шаге. (Подробности опущены). Умножение двух n-битных целых чисел a и b, чтобы использовать другой пример, можно сделать созданием списка n целых чисел длиной не более 2n (каждое из которых равно $a \cdot 2^{i-1} \cdot b^i$, где b^i – i-й бит b) и последующим сложением этих n целых чисел, чтобы получить конечный результат. (Деление с остатком, сложнее для реализации, но также может быть выполнено).

Несмотря на то, что этих алгоритмов начальной школы достаточно, чтобы продемонстрировать, что вышеупомянутые проблемы могут быть решены в течение полиномиального времени, интересно отметить, что эти алгоритмы в некоторых случаях – не самые лучшие из доступных. В качестве примера, простой алгоритм для умножения, приведенный выше, перемножает два n-битных числа за время $O(n^2)$, но существует более хороший алгоритм, действующий в течение времени $O(n^{\log_2 3})$ (и даже этот алгоритм, не является самым лучшим из возможных). В то время как, разница несущественна для такого размера чисел, с которыми мы сталкиваемся ежедневно, она становится заметной, когда числа большие. В криптографических приложениях нередко используются целые числа, длиной тысячи битов (т.е., $n > 1000$) и разумный выбор используемых алгоритмов, становится критическим.

Алгоритм Эвклида и расширенный алгоритм Эвклида

Напомним из раздела 8.1, что gcd(a, b), наибольший общий делитель двух целых чисел a и b – это наибольшее целое число d, которое делит как a, так и b. Сформулируем простое предположение, относящееся к наибольшему общему делителю и затем покажем, как это приводит к эффективному алгоритму его вычисления.

ПРЕДПОЛОЖЕНИЕ В.6 Пусть $a, b > 1$ при $b \mid a$. Тогда gcd(a, b) = gcd(b, [a mod b]).

ДОКАЗАТЕЛЬСТВО Если $b > a$, то актуально заявленное требование. Таким образом, допустим, что $a > b$. Запишем $a = qb + r$ для q, r положительных целых чисел $r < b$ (см. предположение 8.1); заметьте, что $r > 0$ поскольку $b \mid a$. Так как r = [a mod b], докажем это предположение, показав, что gcd(a, b) = gcd(b, r).

Пусть d = gcd(a, b). Тогда d делит как a, так и b, и, таким образом, d делит $r = a - qb$. По определению наибольшего общего делителя мы, таким образом, имеем $\gcd(b, r) \geq d = \gcd(a, b)$.

Пусть dr = gcd(b, r). Тогда dr делит как b , так и r, и, таким образом, dr также делит $a = qb + r$. По определению наибольшего общего делителя мы, таким образом, имеем $\gcd(a, b) \geq dr = \gcd(b, r)$.

Поскольку $d \geq dr$ и $dr \geq d$, мы приходим к выводу, что $d = d^r$.

Указанное выше предполагает рекурсивный алгоритм Эвклида (алгоритм В.7) для вычисления наибольшего общего делителя gcd(a, b) двух целых чисел a и b. Коррект-

ность алгоритма полностью следует из предположения B.6. Что касается длительности его работы, то далее мы покажем, что на входе (a, b) алгоритм делает менее 2 • «b» рекурсивных вызовов. Поскольку проверка, делит ли b a и вычисление

АЛГОРИТМ B.7

Алгоритм Эвклида для наибольшего общего делителя

Вход: Целые числа a, b при $a \geq b > 0$

Выход: Наибольший общий делитель для a и b, если b делит a возвращает b в противном случае, возвращает GCD(b, [a mod b])

o[admb] могут быть сделаны в течение полиномиального времени в «a» и «b», это означает что весь алгоритм действует в течение полиномиального времени.

ПРЕДПОЛОЖЕНИЕ B.8 *Рассмотрим выполнение GCD(a0, b0), и пусть ai, bi (for i = 1, . . . , A) обозначает аргументы для i-го рекурсивного вызова GCD. Тогда* $b_{i+2} \leq b_i/2$ *for* $0 \leq i \leq A - 2$.

ДОКАЗАТЕЛЬСТВО Сначала заметим, что для любого a > b имеем [a mod b] < a/2. Чтобы убедиться в этом, рассмотрим два случая: Если $b \leq a/2$, то [a mod b] < b $\leq$ a/2 является первичным. С другой стороны, если b > a/2, то [a mod b] = a − b < a/2.

Теперь зафиксируем произвольное i при $0 \leq i \leq A - 2$. Тогда $b_{i+2} = [a_{i+1} \bmod b_{i+1}] < a_{i+1}/2 = b_i/2$.

СЛЕДСТВИЕ B.9 В выполнении алгоритма GCD(a, b) существует не более 2 «b» − 2 рекурсивных вызовов для GCD.

ДОКАЗАТЕЛЬСТВО Пусть a_i, b_i (для i = 1, . . . , A) обозначает аргументы для i-го рекурсивного вызова GCD. $\{b_i\}$ всегда больше нуля , алгоритм больше не делает никаких рекурсивных вызовов, если когда-либо случается, что b_i = 1 (поскольку тогда $b_i \mid a_i$). Предыдущее предположение показывает то, что {bi} уменьшается на мультипликативный коэффициент (не менее) 2 за каждые две итерации. Отсюда следует, что количество рекурсивных вызовов для GCD не более 2 • («b» − 1).

Расширенный алгоритм Эвклида

Согласно предположению 8.2, что для положительных целых чисел a, b существуют целые числа X, Y при Xa + Y b = gcd(a, b). Простая модификация алгоритма Эвклида, называемая расширенным алгоритмом Эвклида, может быть использована, чтобы найти X, Y в дополнение к вычислению gcd(a, b); см. алгоритм B.10. Вы просили показать корректность расширенного алгоритма Эвклида в упражнении Exercise B.1 и доказать, что алгоритм действует в течение полиномиального времени в упражнении B.2.

АЛГОРИТМ B.10

Расширенный алгоритм Эвклида eGCD

Вход: Целые числа a, b при $a \geq b > 0$

Выход: (d, X, Y) при d = gcd(a, b) и Xa + Y b = d

if b делит a

return (b, 0, 1)

else

Вычислить целые числа q, r при a = qb + r и $0 < r < b$ (d, X, Y) := eGCD(b, r) // обратите внимание, что Xb + Y r = d return (d, Y, X)

Модульная арифметика

Теперь обратим наше внимание на основные арифметические операции по модулю $N > 1$. Будем использовать ZN, чтобы ссылаться на множество $\{0, \ldots, N - 1\}$, а также на группу, к которой приводит рассмотрение сложения по модулю N среди элементов этого множества.

Основные операции

Эффективные алгоритмы для основных арифметических операций над целыми числами, прямо подразумевают эффективные алгоритмы для соответствующих операций по модулю N . Например, вычисление модульной редукции [a mod N] может быть выполнено в течение полиномиального времени в «a» и «N « при вычислении с остатком по целым числам. Далее рассмотрим модульные операции на двух элементах $a, b \in Z_N$, где «N « = n. (Заметим, что a, b имеют длину не более n. На самом деле, удобно предположить, что все элементы ZN имеют длину точно n, заполняя, при необходимости, левую часть нулями). Сложение a и b по модулю N может быть сделано при первом вычислении a + b, целого числа не более n + 1, а затем редукцией этого промежуточного результата по модулю N . Аналогично, умножение по модулю N может быть выполнено при первом вычислении целого числа по модулю ab длиной не более 2n и затем редукцией результата по модулю N . Поскольку все операции сложения, умножения и деления с остатком могут быть выполнены в течение полиномиального времени, они дают полиномиально-временные алгоритмы для сложения и умножения по модулю N .

Вычисление модульных инверсий

Наше обсуждение к настоящему моменту показало, как сложить, вычесть и умножить по модулю N . Одна операция, которую мы пропустили, – это «деление» или, что то же самое, мультипликативные инверсии по модулю N . Напомним из раздела 8.1.2, что мультипликативная инверсия (по модулю N) элемента $a \in Z_N$ – это элемент $a^{-1} \in Z_N$, такой что, $a \cdot a^{-1} = 1 \bmod N$. Предположение 8.7 показы-

вает, что a имеет инверсию, если и только если $\gcd(a, N) = 1$, т.е., если и только если $a \in Z_N^*$. Таким образом, используя алгоритм Эвклида можно легко определить, имеет ли данный элемент a мультипликативную инверсию по модулю N.

С учетом N и $a \in Z_N$ при $\gcd(a, N) = 1$, предположение 8.2 говорит нам, что существуют целые числа X, Y при $Xa + Y N = 1$. Это означает, что $[X \bmod N]$ является мультипликативной инверсией a. Целые числа X и Y, удовлетворяющие $Xa + Y N = 1$, могут быть эффективно обнаружены с использованием расширенного алгоритма Эвклида eGCD, показанного в разделе B.1.2. Это приводит к следующему полиномиально-временному алгоритму для вычисления мультипликативных инверсий:

АЛГОРИТМ B.11
Вычисление модульных инверсий
Вход: Модуль N ; элемент a
Выход: $[a^{-1} \bmod N]$ (если существует)
(d, X, Y) := eGCD(a, N) // заметьте, что $Xa + Y N = \gcd(a, N)$
if d *f*= 1 return “a не обратимо по модулю N ”

Более сложная задача – это возведение в степень по модулю N, то есть, вычисление $[a^b \bmod N]$ для основания $a \in Z_N$ и степени целого числа $b > 0$. (Когда $b = 0$, задача простая. Если $b < 0$ и $a \in Z_N^*$, то $a^b = (a^{-1})^{-b} \bmod N$ и задача сводится к случаю возведения в степень с положительной степенью с учетом того, что можно вычислить инверсии, как обсуждалось в предыдущем разделе). Обратите внимание, что основной метод, используемый в случае сложения и умножения (т.е., вычисления целого числа ab и последующей редукции этого промежуточного результата по модулю N) здесь не работает: целое число ab имеет длину $a^b = \Theta(\log a^b) = \Theta(b \bullet «a»)$, и даже хранение промежуточного результата a^b потребует времени в степени «b» $= \Theta(\log b)$.

Эту проблему можно решить за счет редукции по модулю N на всех промежуточных этапах вычисления, а не только редукции по модулю N в конце. Это влияет на хранение промежуточных результатов, «небольших» по всему объему вычислений. Даже с этим важным начальным результатом разработка полиномиально-временного алгоритма для модульного возведения в степень, по-прежнему является нетривиальной задачей. Рассмотрим примитивный метод алгоритма B.12, который просто выполняет b умножений на a. Это по-прежнему действует в течение времени, которое является степенью «b». Этот примитивный алгоритм можно рассматривать как основанный на следующем рекуррентном соотношении:

$[a^b \bmod N] = [a \bullet a^{b-1} \bmod N] = [a \bullet a \bullet a^{b-2} \bmod N] = \bullet \bullet \bullet$

Любой алгоритм, основанный на этом соотношении, потребует времени $\Theta(b)$. Мы можем сделать

АЛГОРИТМ B.12
Примитивный алгоритм для модульного возведения в степень
Модуль N ; основание a ∈ Z_N ; Выход: [ab mod N]
x := 1
for i = 1 to b:
x := [x • a mod N

лучше, опираясь на следующее рекуррентное соотношение:

$$[a^b \bmod N] = \begin{cases} \left[\left(a^{\frac{b}{2}}\right)^2 \bmod N\right] & \text{when } b \text{ is even} \\ \left[a \cdot \left(a^{\frac{b-1}{2}}\right)^2 \bmod N\right] & \text{when } b \text{ is odd.} \end{cases}$$

Это приводит к алгоритму, который по очевидным причинам называют «квадрат и умножение» (или «повторяющееся возведение в квадрат»), который требует только O(log b) = O(«b») модульных возведений в квадрат/умножений; см. алгоритм B.13. В этом алгоритме длина b уменьшается на 1 в каждой итерации; отсюда следует, что число итераций равно «b», и поэтому общий алгоритм действует в течение полиномиального времени «a», «b», и «N «. Точнее, количество модульных возведений в квадрат равно точно»b», а количество дополнительных модульных умножений точно равно весу Хэмминга b (т.е., количеству единиц в двоичном представлении

b). Это объясняет предпочтение, обсуждаемое в разделе 8.2.4, по выбору степени e открытого RSA малых длины/веса Хэмминга.

АЛГОРИТМ B.13
Алгоритм ModExp для эффективного модульного возведения в степень Модуль N ; основание a ∈ Z_N ; Выход: [ab mod N]
x := a
t :=1
// поддерживает инвариант, поэтому ответ равен [t • x^b mod N]
while b > 0 do:
if b является четным
t :=[t • x mod N], b := b − 1
x:=[x^2 mod N], b := b/2
retun t

Зафиксируем a и N и рассмотрим функцию модульного возведения в степень с учетом fa,N (b) = [ab mod N]. Мы только что видели, что вычислить $f_{a,N}$ просто. Напротив, вычисление инверсии этой функции, то есть, вычисление b с учетом a, N и [a^b mod N], считается трудным для соответствующего выбора a и N . Инвертирование этой функции требует решения задачи дискретного лога-

рифмирования, кое-что мы обсуждаем подробно в разделе 8.3.2.

Использование предварительных вычислений. Если основание a известно заранее, и существует ограничение на длину показателя степени b, то можно использовать предварительные вычисление и небольшой объем памяти, чтобы ускорить вычисление $[a^b \bmod N]$. Пусть «b» $\leq$ n. Тогда предварительно вычислим и сохраним n значений

$$x_0 := a, \quad x_1 := [a^2 \bmod N], \quad \ldots, \quad x_{n-1} := [a^{2n-1} \bmod N].$$

Принимая во внимание показатель степени b с двоичным представлением $b_{n-1} \bullet\bullet\bullet b_0$ (записанным от наиболее значимого до наименее значимого бита), имеем

$$a^b = a^{\sum_{i=0}^{n-1} 2^i \cdot b_i} = \prod_{i=0}^{n-1} x_i^{b_i} \bmod N.$$

Поскольку $b_i \in \{0, 1\}$, количество умножений, необходимое для вычисления, результат ровно на единицу меньше, чем вес Хэмминга b.

Возведение в степень произвольных групп

Эффективный алгоритм модульного возведения в степень, приведенный выше, переносится в простой способ, чтобы обеспечить эффективное возведение в степень в любой группе, до тех пор, пока операция основной группы может быть выполнена эффективно. В частности, если G является группой, а g – элементом G, то gb можно вычислить, используя не более 2 • «b» применений операции основной группы. Предварительное вычисление может быть использовано в точности так же, как описано выше.

Если порядок q из G известен, то $a^b = a^{[b \bmod q]}$ (см. предположение 8.52) и это можно использовать для ускорения вычисления при редукции b по модулю q, прежде всего.

С учетом (аддитивной) группы ZN , только что описанный алгоритм возведения в степень группы дает метод вычисления «возведения в степень»

$$[b \cdot g \bmod N] \stackrel{\text{def}}{=} [\underbrace{g + \cdots + g}_{b \text{ times}} \bmod N]$$

который отличается от метода, обсуждавшегося ранее, опирающегося на стандартное целочисленное умножение с последующей модульной редукцией. При сравнении двух подходов к решению одной и той же проблемы, обратите внимание, что исходный алгоритм использует специфическую информацию о Z_N ; в частности он (по существу) рассматривает «показатель степени» b в качестве элемента Z_N (возможно, при редукции b по модулю N прежде всего). В противоположность «возведению в квадрат и умножению» только что представленный алгоритм, рассматривает Z_N только как абстрактную группу. (Конечно, групповая операция сложения по модулю N зависит от специфики Z_N). Суть этого обсуждения заключается лишь в том, чтобы показать, что некоторые групповые алгоритмы являются универсальными (т.е., они одинаково хорошо применимы ко всем группам), тогда как некоторые групповые алгоритмы основаны на специфических свойствах отдельных

групп или классов групп. Мы видели некоторые примеры этого явления в главе 9.

***Умножение Монтгомери**

Хотя деление целых чисел (и, следовательно, модульная редукция) могут быть выполнены в течение полиномиального времени, алгоритмы для целочисленного деления медленные, в сравнении, например, с алгоритмами для целочисленного умножения. Умножение Монтгомери дает способ выполнения модульного умножения без выполнения каких-либо затратных модульных редукций. Поскольку требуется предварительная и заключительная обработка, метод выгоден только тогда, когда несколько модульных умножений будет сделано последовательно, как, например, при вычислении модульного возведения в степень.

Зафиксируем нечетный модуль N, по отношению к которому должны быть выполнены модульные операции. Пусть R > N будет степенью двух, скажем $R = 2^w$, и заметьте, что gcd(R, N) = 1. Ключевое свойство, которое мы будет использовать состоит в том, что деление на R – быстрое: коэффициент x при делении на R получается при простом сдвиге x вправо w позиций, и [x mod R] является только w битов с наименьшими значениями x.

Определим представление Монтгомери $x \in Z_N^*$ с помощью $\bar{x} \stackrel{def}{=} [xR \bmod N]$. Умножение Монтгомери $\bar{x}, \bar{y} \in Z_N^*$ определяется как

$$\text{Mont}(\bar{x}, \bar{y}) \stackrel{def}{=} [\bar{x}\bar{y}R^{-1} \bmod N].$$

(Ниже мы покажем, как это можно вычислить без каких-либо затратных модульных редукций). Заметим, что

$$\text{Mont}(\bar{x}, \bar{y}) = \bar{x}\bar{y}R^{-1} = (xR)(yR)R^{-1} = (xy)R = \text{oxdy m} \qquad N.$$

Это означает, что можно умножить несколько значений в ZN посредством: (1) преобразования в представление Монтгомери, (2) выполнением всех умножений с использованием умножения Монтгомери, чтобы получить конечный результат, а затем (3) преобразования результата из представления Монтгомери, назад к стандартному представлению. Пусть $\alpha \stackrel{def}{=} [-N^{-1} \bmod R]$ – значение, которое может быть предварительно вычислено. (Вычисление α, преобразование в представление Монтгомери/из него, также может быть выполнено без каких-либо затратных модульных редукций; подробности не входят в сферу наших интересов). Чтобы вычислить $c \stackrel{def}{=}$ Mont(x, y) без каких-либо затратных модульных редукций сделаем следующее:

1. Пусть z := x • y (по целым числам).
2. Зададим $c^r := (z + [z\alpha \bmod R] \bullet N) / R$.
3. Если $c^r < N$, то зададим $c := c^r$; иначе $c := c^r - N$.

Чтобы убедиться в работоспособности этого, сначала нужно проверить, что шаг 2 правильно определен, а именно, что числитель делится на R. Это следует из того, что

$$z + [z\alpha \bmod R] \bullet N = z + z\alpha N = z - zN^{-1}N = 0 \bmod R.$$

Далее обратите внимание, что cr = z/R mod N после шага 2; кроме того, поскольку $z < N^2 < RN$, мы имеем $0 < c^r < (z + RN)/R < 2RN/R = 2N$. Но тогда $[cr \bmod N] = c^r$ если $c^r < N$, и $[c^r \bmod N] = cr - N$ if $c^r > N$. Мы приходим к выводу, что

$$c = [c^r \bmod N] = [z/R \bmod N] = [xyR^{-1} \bmod N],$$

что и требовалось доказать.

Выбор однородного группового элемента

Для криптографических приложений часто бывает необходимо выбрать однородный элемент группы G. Сначала рассмотрим проблему в абстрактной обстановке, а затем сосредоточимся именно на случаях Z_N and Z_N^* .

Заметим, что если G является циклической группой порядка q, и генератор $g \in G$ известен, то выбор однородного элемента $h \in G$ сводится к выбору однородного целого $x \in Zq$ и заданию $h := g^x$. В дальнейшем мы не делаем никаких допущений относительно G.

Элементы группы G должны быть указаны с использованием некоторого представления этих элементов в качестве строк битов, где мы предполагаем, без какой-либо потери общности, что все представленные элементы используют строки одной и той же длины. (Важно также, что существует единственная строка, представляющая каждый групповой элемент). Например, если $\|N\| = n$, то все элементы Z_N могут быть представлены в виде строк n, где целое число $a \in Z_N$ заполняется налево нулями, если $\|a\| < n$.

Мы не уделяем много внимания вопросу о представлении, поскольку для всех групп, рассматриваемых в этом тексте, представление может быть принято как «естественное» представление (как в случае Z_N , приведенном выше). Однако заметим, что различные представления одной и той же группы, могут влиять на сложность выполнения различных вычислений, и поэтому выбор «правильного» представления для данной группы, зачастую важен на практике. Поскольку наша задача заключается в том, чтобы показать полиномиально-временные алгоритмы для каждой из операций (и не показывать наиболее эффективные известные алгоритмы), то используемое точное представление менее важно для нас. Более того, большинство алгоритмов «более высокого уровня», которые мы представляем, используют групповую операцию в стиле «черного ящика», таким образом, что до тех пор, пока групповая операция может быть выполнена в течение полиномиального времени (по какому-то параметру), результирующий алгоритм также будет действовать в течение полиномиального времени.

Учитывая группу G, где элементы представлены строками длиной A, однородный элемент группы можно отобрать, выбирая равномерные A-битные строки до тех пор, пока не найдена первая строка, соответствующая элементу группы. (При этом предполагается, что тестирование члена группы может быть выполнено эффективно). Чтобы получить алгоритм с ограниченным временем действия, введем параметр t, ограничивающий максимальное количество повторений этого процесса; если после

всех t итераций не удалось найти элемент G, то алгоритм выводит fail (ошибка). (Альтернативой будет вывод произвольного элемента G). Другими словами:

АЛГОРИТМ В.14
Выбор элемента однородной группы
Вход: (Описание) группы G; параметр длины A; параметр t
Выход: Однородный элемент группы G
for i = 1 to t:
Выбрать однородный if x ∈ G return x
return "fail"

Очевидно, что всякий раз, когда указанный выше алгоритм не выводит fail (ошибка), он выводит равномерно распределенный элемент G. Это просто потому, что каждый элемент G с равной вероятностью может быть выбран в любой итерации. С формальной точки зрения, если допустить, чтобы Fail было событием, при котором алгоритм выводит fail (ошибка), то для любого элемента g∈ G имеем

Pr[выход алгоритма равен g | Fail] =1/|G|

Какова вероятность того, что алгоритм выводит fail (ошибка)? При любой итерации, в ероятность того, что x ∈ G равно точно $|G|/2^A$, и вероятность того, что x не лежит в G при любой из t итераций равна

$$\left(1 - \frac{|\mathbb{G}|}{2^\ell}\right)^t . \qquad \text{(B.1)}$$

Существует компромисс между временем работы алгоритма B.14 и вероятностью того, что алгоритм выводит fail (ошибка): увеличение t уменьшает вероятность ошибки, но увеличивает время работы в худшем случае. Для криптографических приложений нам нужен алгоритм, при котором время работы в худшем случае было бы полиномом в параметре безопасности n, тогда как вероятность ошибки пренебрежимо мала в n. Пусть $K \stackrel{\text{def}}{=} 2^A/|G|$. Если задать t := K • n, то вероятность того, что алгоритм выводит fail (ошибка) равна:

$$\left(1 - \frac{1}{K}\right)^{K \cdot n} = \left(\left(1 - \frac{1}{K}\right)^K\right)^n \leq \left(e^{-1}\right)^n = e^{-n},$$

при использовании предположения A.2. Таким образом K = poly(n) (мы предполагаем некий алгоритм групповой генерации, который зависит от параметра безопасности n, и, таким образом, как |G|, так и A являются функциями n), мы получаем алгоритм с нужными свойствами.

Случай Z_N . Рассмотрим группу Z_N , при n = ||N||. Проверка, является ли n-битная строка x (интерпретируется как положительное целое число длиной не более n) элементом Z_N, просто требует проверки, что x < N . Кроме того,

$$\frac{2^n}{|\mathbb{Z}_N|} = \frac{2^n}{N} \leq \frac{2^n}{2^{n-1}} = 2$$

и поэтому мы может отобрать однородный элемент Z_N в течение времени poly(n) и с пренебрежимо малой вероятностью ошибки в n.

Случай Z_N^* . Рассмотрим следующую группу Z_N^* , при n = ||N|| , как и ранее. Определить, является ли n-битная строка x элементом Z_N^* так же просто (см.пражнения). Более того,

$$\frac{2^n}{|\mathbb{Z}_N^*|} = \frac{2^n}{\phi(N)} = \frac{2^n}{N} \cdot \frac{N}{\phi(N)} \leq 2 \cdot \frac{N}{\phi(N)} .$$

Верхняя граница poly(n) является следствие приведенной ниже теоремы.

ТЕОРЕМА B.15 *Для $N \geq 3$ длиной n, имеем $N/\varphi(N) < 2n$.*

(Известны более строгие ограничения, но для нашей цели достаточно указанных выше). Теорему можно доказать с помощью постулата Бертрана (теорема 8.32), но мы ограничимся доказательством двух особых случаев: когда N является простым числом и когда N является произведением двух (различных) простых чисел одинаковой длины.

Анализ прост, когда N – четное число. Здесь $\varphi(N) = N - 1$, и таким образом

$$\frac{N}{\phi(N)} \leq \frac{2^n}{\phi(N)} = \frac{2^n}{N-1} \leq \frac{2^n}{2^{n-1}} = 2$$

(используя тот факт, что N является нечетным для второго неравенства). Рассмотри следующий случай N = pq для p и q различных, нечетных простых чисел. Тогда

$$\frac{N}{\phi(N)} = \frac{pq}{(p-1)(q-1)} = \frac{p}{p-1} \cdot \frac{q}{q-1} < \left(\frac{3}{2}\right) \cdot \left(\frac{5}{4}\right) < 2.$$

Мы пришли к выводу, что когда N является простым или произведением двух различных нечетных простых чисел, существует алгоритм для генерации однородного элемента Z_N^*, который действует в течение времени полинома в n = ||N|| и выводит fail (ошибка) с вероятностью пренебрежимо малой в n.

На протяжении этой книги, когда мы говорим об отборе однородного элемента из Z_N или Z_N^* мы попросту игнорируем пренебрежимо малую вероятность вывода fail (ошибка), понимая, что это не оказывает существенного влияния на анализ.

***Нахождение генератора циклической группы**

В этом разделе мы рассмотрим задачу нахождения генератора произвольной циклической группы G , порядка q. Здесь q не обязательно обозначает простое число; на самом деле, нахождение генератора когда q – тривиально согласно следствию 8.55.

Сейчас мы покажем, как отобрать равномерный генератор, действуя способом, очень похожим на способ из раздела B.2.5. Здесь мы неоднократно отбираем однородные элементы G до тех пор, пока не найдем элемент, который является генера-

тором. Как и в разделе B.2.5, анализ этого метода требует понимания двух вещей:

• Как эффективно проверить, является ли данный элемент генератором, а также

• проверить часть элементов группы, которые являются генераторами.

Чтобы понять эти вопросы, сначала разработаем небольшое дополнительное теоретически-групповое основание.

Теоретико-групповое основание

Сначала займемся вторым вопросом. Напомним, что порядок элемента h – это наименьшее целое число i, для которого $h^i=1$. Пусть g будет генератором группы G порядка $q > 1$; это означает, что порядок g равен q. Рассмотрим элемент $h \in G$, который не является единичным (единичный элемент не может быть генератором G), и проверим h, может ли также быть генератором G. Поскольку g генерирует G, можно записать $h = g^x$ для некоторого $x \in \{1, \ldots, q-1\}$ (обратите внимание, что $x \neq 0$, поскольку h не является единичным элементом). Рассмотрим два случая:

Случай 1: $\gcd(x, q) = r > 1$. Запишем $x = \alpha \cdot r$ и $q = \beta \cdot r$ при этом α, β – ненулевые целые числа меньше q. Тогда:

$h^\beta = (g^x)^\beta = g^{\alpha r \beta} = (g^q)^\alpha = 1$.

Таким образом, порядок h не более $\beta < q$ и h не может быть генератором G.

Случай 2: $\gcd(x, q) = 1$. Пусть $i \leq q$ будет порядком h. Тогда

$g^0 = 1 = h^i = (g^x)^i = g^{xi}$,

подразумевая, что $xi = 0 \bmod q$ согласно предположению 8.53. Это означает, что $q \mid x^i$. Поскольку $\gcd(x, q) = 1$, тем не менее, предположение 8.3 показывает, что $q \mid i$, и, таким образом, $i = q$. Мы пришли к выводу, что h является генератором G.

Резюмируя сказанное выше, видим, что для $x \in \{1, \ldots, q-1\}$ элемент $h = g^x$ точно является генератором G. когда $\gcd(x, q) = 1$. Итак, мы доказали следующее:

ТЕОРЕМА B.16 *Пусть G – циклическая группа порядка $q > 1$ с генератором g. Существует $\varphi(q)$ генераторов G и они точно задаются $\{g^x \mid x \in Zg^*\}$.*

В частности, если G является группой порядка простого числа q, то она имеет $\varphi(q) = q - 1$ генераторов—точно в соответствии со следствием 8.55.

Теперь мы вернемся к первому вопросу – вопросу принятия решения, является ли данный элемент h генератором G. Конечно, один способ проверить, генерирует ли h G – это перечислить $\{h_0, h_1, \ldots, h_{q-1}\}$ и посмотреть, включает ли этот список каждый элемент G. Это требует линейного времени в q (т.е., экспоненциального в $\|q\|$) и поэтому неприемлемо для наших целей. Другой подход, если мы уже знаем генератор g, заключается в том, чтобы вычислить дискретный логарифм $x = \log_g h$ и затем применить предыдущую теорему; однако в общем случае, мы можем не иметь такого g, и, в любом, случае само вычисление дискретного логарифма может быть сложной задачей.

Если нам известна факторизация (простые множители) q, то мы можем поступить лучше.

PROPOSITION B.17 *Пусть G будет группой порядка q, а* $q = \prod_{i=1}^{k} p_i^{e_i}$ *разложением на простые множители q, где {pi} – различные целые числа и ei ≥ 1. Зададим qi = q/pi. Тогда h ∈ G является генератором G , если и только если*

$h^{q_i} \quad f = 1 \quad \text{for } i = 1, \ldots, k.$

ДОКАЗАТЕЛЬСТВО Одно из направлений простое. Допустим, что $h^{q_i} = 1$ для некоторого i. Тогда порядок h не более qi < q, и, таким образом, h не может быть генератором.

С другой стороны, пусть h не является генератором, но вместо этого имеет порядок qr < q. Согласно предположению 8.54, мы знаем, что qr q. Это подразумевает, что qr можно записать как $q^r = \prod_{i=1}^{k} p_i^{e_i'}$, где $e_i' \geq 0$ и, минимум, для одного индекса j мы имеем $e_i' < e_j$. Но тогда q^r делит $q_j = p_j^{e_j - 1} \cdot \prod_{i \neq j} p_i^{e_i}$, и, таким образом, (используя предположение 8.53) $h^{q_j} = h^{[q_j \bmod q]} = h^0 = 1$.

Это предположение не требует, чтобы G была циклической; если G не является циклической, то каждый элемент h ∈ G будет удовлетворять $h^{q_i} = 1$ для некоторого i и генераторы не существуют.

Эффективные алгоритмы

Имея результаты предыдущего раздела, покажем, как эффективно проверить, является ли данный элемент генератором, а также, как эффективно найти генератор в произвольной группе.

Проверка, что элемент является генератором. Предположение B.17, прямо предлагает эффективный алгоритм для решения вопроса, является ли данный элемент h генератором или нет.

АЛГОРИТМ B.18

Проверка, что элемент является.

Вход: Групповой порядок q; $\{p^i\}_{i=1}^{k}$ q; элемент h ∈G Выход: Решение относительно того, является ли h генератором G

```
for i = 1 to k:
if h^(q/pi) = 1 return "h не является генератором"
return "h является генератором"
```

Корректность алгоритма очевидна из предположения B.17. Теперь покажем, что алгоритм завершает работу в течение полиномиального времени в «q». Поскольку в каждой итерации h^{q/p_i} можно вычислить в течение полиномиального времени, нам нужно только показать, что количество итераций k является полиномом. Это как раз тот случай, поскольку целое число q может иметь не более, чем $\log_2 q = O(\|q\|)$ простых множителей; это потому, что

$$q = \prod_{i=1}^{k} p_i^{e_i} \geq \prod_{i=1}^{k} p_i \geq \prod_{i=1}^{k} 2 = 2^k$$

и, таким образом, $k \leq \log_2 q$.

Алгоритм B.18 требует, чтобы простые множители порядка группы q были предоставлены в виде входных данных. Интересно заметить, что не известен эффективный алгоритм для проверки, является ли элемент произвольной группы генератором, когда коэффициенты группового порядка не известны.

Часть элементов, которые являются генераторами. Как показано в теореме B.16, часть элементов группы G порядка q, которые являются генераторами, это $\varphi(q)/q$. Теорема B.15 гласит, что $\varphi(q)/q = \Omega(1/\|q\|)$. Доля элементов, которые являются генераторами, таким образом, достаточно высока, чтобы обеспечить, что отбор полиномиального количества элементов из группы даст генератор с практически пренебрежимо малой вероятностью. (Анализ тот же самый, как и в разделе B.2.5).

Конкретные примеры Z_p^*. Сопоставив все, мы видим, что существует эффективный вероятностный алгоритм для нахождения генератора группы G до тех пор, пока факторизация порядка группы известна. Поэтому, при выборе группы для криптографических приложений важно, чтобы группа выбиралась таким образом, чтобы это было справедливым. Это опять же объясняет предпочтение, которое подробно обсуждалось в разделе 8.3.2, для работы в соответствующей подгруппе Z_p^* порядка простого числа. Другая возможность заключается в том, чтобы использовать $G = Z_p^*$ для p сильного p рпростого числа (т.е., $p = 2q + 1$ при q также простом), и в этом случае простые множители группового порядка $p - 1$ известны. Одна последняя возможность заключается в том, чтобы генерировать простое число p таким образом, что факторизация $p - 1$ известна. Дальнейшие подробности выходят за пределы данной книги.

Ссылки и дополнительная литература

Книгу Шоупа (Shoup) настоятельно рекомендуем тем, кто стремится изучить темы этой главы более подробно. В частности, ограничения на $\varphi(N)/N$ (и асимптотическую версию теоремы B.15) можно найти в [159, глава 5]. Ханкерсон и др. (Hankerson et al.) [83] также предоставляют обширные подробности о реализации теоретико-числовых алгоритмов для криптографии.

Упражнения

Докажите корректность расширенного алгоритма Эвклида.

Докажите, что расширенный алгоритм Эвклида работает в течение полиномиального времени, на длине его входных данных.

> **Подсказка:** Сначала докажем предположение, аналогичное предположению B.8.

Покажите, как определить, что n-битная строка находится в Z_N^* в течение полиномиального времени.

Содержание

3.5 Создание CPA-устойчивых систем шифрования
    3.5.1 Псевдослучайные функции и блочные шифры
    3.5.2 Шифрование, защищенное от атак с выбором открытого текста, из псевдослучайных функций
3.6 Режимы использования
    3.6.1 Режимы использования с поточными шифрами
    3.6.2 Режимы использования с блочными шифрами
3.7 Атаки с выбором шифртекста
    3.7.1 Определение устойчивости против АВШ
    3.7.2 Атаки оракула дополнения

4 ГЛАВА. Коды аутентификации сообщений
4.1 Целостность сообщения
    4.1.1 Секретность и целостность
    4.1.2 Шифрование и аутентификация сообщений
4.2 Коды аутентификации сообщений – определения
4.3 Построение безопасных кодов аутентификации сообщений
    4.3.1 КАС с фиксированной длиной
    4.3.2 Расширение области для кодов аутентификации сообщений
4.4 CBC-MAC
    4.4.1 Базовая конструкция
    4.4.2 *Доказательство беопасности
4.5 Аутентифицированное шифрование
    4.5.1 Определения
    4.5.2 Обобщенные конструкции
    4.5.3 Защищенные сеансы связи
    4.5.4 Шифрование против атак на основе подобранного шифротекста
4.6 Инфрмационно-теоретические MAC
    4.6.1 Построение информационно-теоретических MAC
    4.6.2 Ограничение информационно-теоретических MAC
Ссылки и дополнительная литература
Упражнения

5 ГЛАВА. Хэш-функции и их применения

5.1 Определения
    5.1.1 Стойкость к коллизиям
    5.1.2 Более слабые понятия о защите
5.2 Расширение области: Преобразование Меркле-Дамгорда