Б.А. Фороузан

Математика криптографии и теория шифрования

ИНТУИТ

НАЦИОНАЛЬНЫЙ ОТКРЫТЫЙ УНИВЕРСИТЕТ

Математика криптографии и теория шифрования

2-е издание, исправленное

Фороузан Б.А.

Национальный Открытый Университет "ИНТУИТ"
2016

УДК 003.26(075.8)
ББК 34
Ф79
Криптография и безопасность сетей / Фороузан Б.А. - М.: Национальный Открытый Университет "ИНТУИТ", 2016 (Основы информационных технологий)
ISBN 978-5-9963-0242-0

Данный курс дает представление о криптографии с симметричными ключами (симметричное шифрование) - и шифровании с помощью асимметричных ключей (асимметричное шифрование).

В первой части курса описывается как традиционный, так и современный подход к процессу шифрования симметричными ключами. Лекции второй части показывают, как асимметричное шифрование может обеспечить безопасность информации.

Введение

Эта лекция преследует несколько целей: определить три цели информационной безопасности, определить виды атак на безопасность информации, которые угрожают секретности, определить службы безопасности и как они связаны с тремя задачами безопасности, определить механизмы безопасности, которые обеспечивают службы секретности, познакомить с двумя методами шифрования для реализации механизма безопасности — криптографией и стеганографией.

Мы живем в информационную эпоху. Мы должны внимательно сохранять информацию о каждом аспекте нашей жизни. Другими словами, информация — собственность, и, подобно любой другой собственности, имеет важное значение. И в этом качестве информация должна быть защищена от нападений.

Информация должна быть сохранена от неправомочного доступа (конфиденциальность), защищена от неправомочного изменения (целостность) и доступна только разрешенному объекту, когда это ему необходимо (доступность).

Несколько десятилетий назад информация собиралась на физических носителях. Конфиденциальность этих носителелй достигалась строгим ограничением доступа, который предоставлялся только людям, имеющим на это право, и тем из них, кому можно было доверить эту информацию. Также нескольким правомочным субъектам разрешалось изменение содержания этих файлов. Готовность была обеспечена тем, что по меньшей мере одному человеку всегда разрешался доступ к этим носителям.

С появлением компьютеров хранение информации стало электронным. Информация хранилась уже не в физической неэлектронной среде — она накапливалась в электронной среде (компьютерах). Однако три требования безопасности не изменились. Файлы, записанные в компьютере, требовали конфиденциальности, целостности и доступности. Реализация этих требований возможна различными методами и требует решения сложных задач.

В течение прошлых двух десятилетий компьютерные сети произвели революцию в использовании информации. Информация теперь распределена. Люди при наличии полномочий могут передавать информацию и искать ее на расстоянии, используя компьютерные сети. Но три уже упомянутых требования — конфиденциальности, целостности и доступности — не изменились. Они лишь приобрели некоторые новые аспекты. Теперь недостаточно того, что информация должна быть конфиденциальной, когда она сохраняется в компьютере. Должен также существовать способ поддержки конфиденциальности, когда эта информация передается от одного компьютера к другому.

В этой лекции мы сначала обсуждаем три главных цели поддержки безопасности информации.

Когда будет понятно, какие атаки могут угрожать этим трем целям, тогда можно обсудить службы безопасности, предназначенные для этих целей. Потом определяются механизмы обеспечения службы безопасности и методы, которые могут использоваться, чтобы осуществить эти механизмы.

1.1. Цели поддержки безопасности

Рассмотрим три цели поддержки информационной безопасности: конфиденциальность, целостность и готовность.

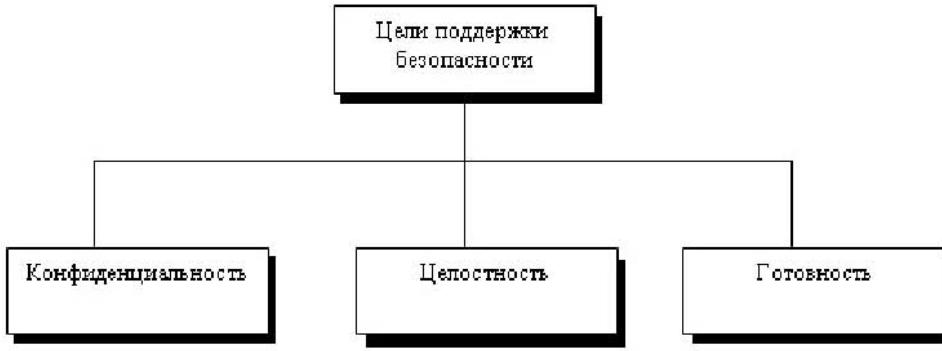


Рис. 1.1. Систематизация целей поддержки безопасности

Конфиденциальность

Конфиденциальность — вероятно, самый общий аспект информационной безопасности. Мы должны защитить нашу конфиденциальную информацию. Организация должна принять меры против тех злонамеренных действий, которые могут нарушить конфиденциальность информации. В военных организациях сохранение секретности важной информации – главная забота руководства. В промышленности сохранение тайны некоторой информации от конкурентов является одним из основных факторов работы организации. В банковском деле должна сохраняться секретность учетных записей клиентов.

Как мы увидим позже в этой лекции, конфиденциальность нужна не только при хранении информации, она также необходима при ее передаче. Когда мы передаем часть информации, которая должна будет храниться в удаленном компьютере, или когда мы отыскиваем информацию, которая находится в удаленном компьютере, мы должны гарантировать ее секретность в течение передачи.

Целостность

Потребители изменяют информацию постоянно. В банке, когда клиент вносит или снимает деньги, баланс на его счету должен быть изменен. Целостность означает, что изменения должны быть сделаны только разрешенными объектами и с помощью разрешенных механизмов. Нарушение целостности — не обязательно результат злонамеренного действия; сбой в системе, такой, например, как всплеск или прерывание мощности в первичной сети электропитания, может привести к нежелательным изменениям некоторой информации.

Готовность

Третий компонент информационной безопасности — готовность. Информация, созданная и сохраненная организацией, должна быть доступна разрешенным объектам. Информация бесполезна, если она не доступна. Информация должна постоянно изменяться, и поэтому тоже должна быть доступна для разрешенных объектов. Неготовность информации столь же вредна для организации, как отсутствие

конфиденциальности или целостности. Вообразите, что случилось бы с банком, если клиенты не могли бы обратиться к своим счетам для снятия или вклада денег.

1.2. Атаки

Нашим трем целям информационной безопасности — конфиденциальности, целостности и готовности — могут угрожать атаки с целью нарушения безопасности информации. Хотя в литературе встречаются различные подходы к систематизации атак, мы сначала разделим их на три группы, связанные с целями нарушения информационной безопасности. Позже мы будем делить их на две широких категории, основанные на эффективности их воздействия на систему. Рисунок 1.2 показывает первую систематизацию.

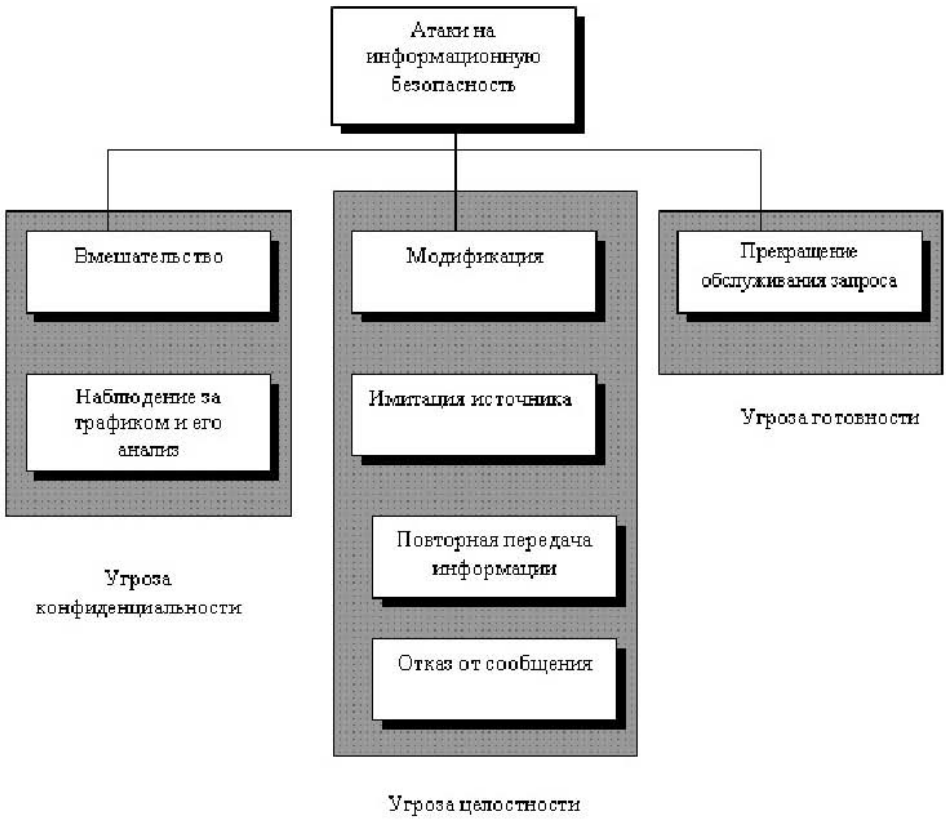


Рис. 1.2. Систематизация атак и соотношение их с целями поддержки

Атаки, угрожающие конфиденциальности

Вообще, имеется два типа атак, которые угрожают конфиденциальности информации: вмешательство и наблюдение за трафиком и его анализ.

Вмешательство

Вмешательство относится к неправомочному доступу или перехвату данных. Например, файл, передаваемый через Интернет, может содержать конфиденциальную информацию. Объект, не имеющий полномочий, может прервать передачу и использовать передаваемую информацию для собственной выгоды. Чтобы предотвратить вмешательство, данные могут быть представлены так, что перехвативший не сможет понять их. Для этого применимы методы шифрования, приводимые в этом курсе.

Наблюдение за трафиком и его анализ

Хотя шифрование данных может сделать их непонятными для злоумышленника, он, анализируя сетевой трафик, может получить некоторую другую информацию. Например, он может найти электронный адрес (адрес электронной почты) передатчика или приемника. Он может также собрать пары запросов и ответов, что поможет ему понять активность действий наблюдаемой организации.

Атаки, угрожающие целостности

Целостности данных можно угрожать несколькими видами атак, такими как модификация, имитация источника, повторная передача информации и отказ от сообщения.

Модификация

После прерывания или доступа к информации атакующий изменяет информацию с определенной выгодой для себя. Например, клиент передает сообщение банку, чтобы провести некоторую операцию. Атакующий прерывает сообщение и изменяет тип операции, чтобы принести этим пользу себе. Обратите внимание, что иногда атакующий просто удаляет или задерживает сообщение, чтобы навредить системе или извлечь выгоду из самого факта задержки операции.

Имитация источника

Имитация источника (spoofing) заключается в том, что атакующий имитирует кого-то, кто имеет право на производимые действия. Например, атакующий захватывает банковскую кредитную карточку и PIN-код клиента банка. Он может действовать как настоящий клиент. Иногда атакующий имитирует приемник. Например, пользователь хочет войти в контакт с банком, но ему предоставляется другой сайт, который имитирует, что это – банк, и атакующий получает необходимую ему информацию от пользователя.

Повторная передача информации

Другой вид атаки — повторная передача информации (атака воспроизведения). Атакующий получает копию сообщения, передаваемого пользователем, и передает эту копию с целью дезорганизации процесса или попыток повторить его. Например, человек передает запрос банку, чтобы оплатить работу своему сотруднику. Атакующий перехватывает сообщение и передает его снова, чтобы получить оплату этой работы от этого банка еще раз (повторно).

Отказ от сообщения

Этот тип атак отличается от других, потому что это действие может быть выполнено одной из двух сторон связи: передатчиком или приемником. Передатчик сообщения может отказаться и отрицать факт передачи сообщения. Другой вариант — когда приемник сообщения отрицает, что он получил сообщение. Пример отказа от сообщения

передатчика: клиент банка передал запрос банку на перевод некоторой суммы денег третьему лицу, но впоследствии отрицает, что он сделал такой запрос. Пример опровержения сообщения приемником: человек, который покупает изделие у изготовителя и платит за это с помощью электроники, но изготовитель позже отрицает, что получил оплату, и просит оплатить покупку.

Атаки, угрожающие готовности

Рассмотрим только одну атаку, угрожающую готовности: отказ в обслуживании.

Отказ в обслуживании

Отказ в обслуживании (Denial of Service — DoS) — очень общее название атаки. Она может замедлить или полностью прервать обслуживание системы. Атакующий может использовать несколько стратегий, чтобы достигнуть этого. Атакующий может передать так много фиктивных запросов серверу, что это приведет к сбою сервера из-за высокой нагрузки. Атакующий может также прервать и удалить ответ сервера клиенту, порождая у клиента впечатление, что сервер не отвечает. Атакующий может прервать запросы от клиентов, порождая у клиента уверенность, что сервер не отвечает. Атакующий может прерывать запросы клиентов, заставляя клиентов передать запросы много раз и перезагружать систему.

Пассивные и активные атаки

Давайте теперь разделим атаки на две группы: пассивные и активные. Таблица 1.1 показывает соотношения между этим делением и предыдущей классификацией.

Пассивные нападения

При пассивном нападении цель атакующего состоит в том, чтобы только

получить информацию. Это означает, что нападение не изменяет данные и не повреждает систему. Система продолжает нормально работать. Однако атака может нанести вред передатчику или приемнику сообщения. Атаки, которые угрожают конфиденциальности — вмешательство и наблюдение за трафиком плюс его анализ, — являются пассивными. Раскрытие информации может вредить передатчику или приемнику сообщения, но систему не затрагивает. По этой причине трудно обнаружить этот тип нападения, пока передатчик или приемник не узнают об утечке конфиденциальной информации. Пассивные нападения, однако, могут быть предотвращены шифрованием данных.

Таблица 1.1. Классификация пассивных и активных атак

Атаки	Пассивные / Активные	Угроза
Вмешательство	Пассивные	Конфиденциальности
Наблюдение за трафиком и его анализ		
Модификация	Активные	Целостности
Имитация источника		
Повторная передача информации		
Отказ от сообщения		
Отказ в обслуживании	Активные	Готовности

Активные атаки

Активные атаки изменяют данные или повреждают систему. Атаки, которые угрожают целостности или готовности, — активные. Активные атаки обычно легче обнаруживаются, чем предотвращаются, потому что атакующий может начинать их разнообразными методами.

1.3. Услуги и механизмы

Международный Союз Электросвязи, Секция Стандартов по телекоммуникации (ITU-T) (см. приложение B) разработал стандарты

некоторых служб безопасности и некоторые механизмы для осуществления этих услуг. Службы информационной безопасности и механизмы близко связаны, потому что механизм или комбинация механизмов применяются, чтобы обеспечить обслуживание. Механизм может использоваться в одной или нескольких услугах. Ниже эти механизмы кратко обсуждаются, чтобы понять их общую идею. Далее они будут рассмотрены более подробно.

Услуги информационной безопасности

ITU-T (X.800) определил пять услуг, связанных с целями информационной безопасности и атаками, типы которых мы определили в предыдущих секциях. Рисунок 1.3 показывает классификацию пяти общих услуг.

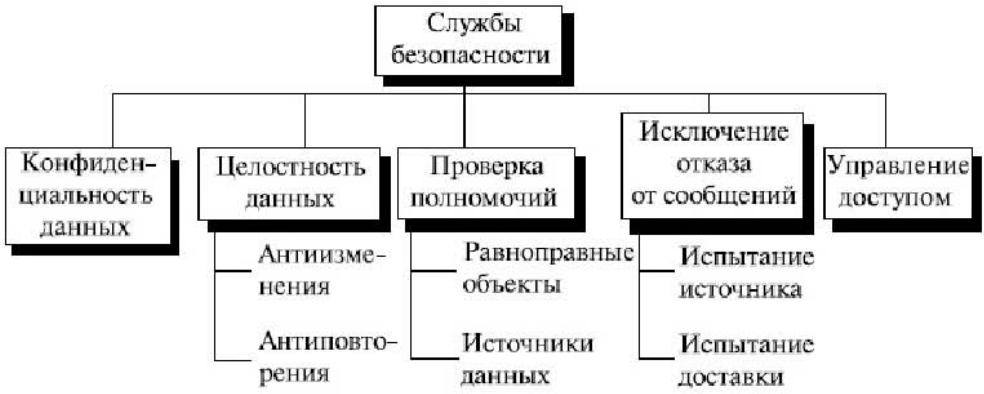


Рис. 1.3. Услуги информационной безопасности

Чтобы предотвратить атаки на информационную безопасность, о которых мы говорили, надо просто иметь одну или больше показанных выше услуг для одного или большего количества целей информационной безопасности.

Конфиденциальность данных

Конфиденциальность данных разработана, чтобы защитить данные от попытки их раскрытия. Эта широкая услуга, определенная в

рекомендации ITU-T X.800. Она может охватывать конфиденциальность целого сообщения или его части, а также защищает от наблюдения за трафиком и его анализа — собственно, она разработана для предотвращения вмешательства и наблюдения за трафиком.

Целостность данных

Целостность данных разработана для защиты данных от модификации, вставки, удаления и повторной передачи информации противником. Она может защищать целое сообщение или часть сообщения.

Установление подлинности (аутентификация)

Эта услуга обеспечивает установление подлинности (аутентификацию) оператора на другом конце линии. При соединении, ориентированном на подключение, она обеспечивает установление подлинности передатчика или приемника в течение установления соединения (установление подлинности объектов равного уровня). При соединении без установления подключения она подтверждает подлинность источника данных (установление подлинности происхождения данных).

Исключение отказа от сообщений

Услуга исключение отказа от сообщений защищает от отказа от сообщения передатчиком или приемником данных. При исключении отказа от сообщения передатчиком приемник данных может потом доказать происхождение сообщения, используя опознавательный код (идентификатор) передатчика. При исключении отказа от сообщений приемником передатчик, используя подтверждение доставки, может потом доказать, что данные доставлены предназначенному получателю.

Управление доступом

Управление доступом обеспечивает защиту против неправомочного доступа к данным. Доступ в этом определении — термин очень

широкий и может включать чтение, запись, изменение данных, запуск выполнения программы и так далее.

Механизмы безопасности

Для обеспечения услуг информационной безопасности ITU-T (X.800) рекомендует некоторые механизмы безопасности, определенные в предыдущей секции. Рисунок 1.4 дает классификацию этих механизмов.

Рис. 1.4. Механизмы информационной безопасности

Шифрование

Шифрование. Засекречивая или рассекречивая данные, можно обеспечить конфиденциальность. Шифрование также дополняет другие механизмы, которые обеспечивают другие услуги. Сегодня для шифрования используются два метода: криптография и стеганография — тайнопись (steganography). Мы коротко обсудим их в дальнейшем.

Целостность данных

Механизм целостности данных добавляет в конце данных короткий контрольный признак (check value), который создается определенным процессом отдельно от данных. Приемник получает данные и контрольный признак. На основании полученных данных он создает новый контрольный признак и сравнивает только что созданный с полученным. Если эти два контрольных признака совпадают, целостность данных была сохранена.

Цифровая подпись

Цифровая подпись — средство, которым отправитель может с помощью электроники подписать данные, а приемник может с помощью компьютера проверить подпись. Отправитель использует процесс, который может указать, что эта подпись имеет частный ключ, выбранный из общедоступных ключей, которые были объявлены публично для общего пользования. Приемник использует общедоступный ключ отправителя, чтобы доказать, что сообщение действительно подписано отправителем, который утверждает, что послал сообщение.

Обмен сообщениями для опознавания

При обмене сообщениями для опознавания два объекта обмениваются некоторыми сообщениями, чтобы доказать, что эти объекты известны друг другу. Например, одно юридическое лицо может доказать, что оно знает тайный признак, который только оно может знать (скажем, последнее место встречи с партнером).

Заполнение трафика

Заполнение трафика означает возможность вставлять в трафик данных некоторые фиктивные данные, чтобы сорвать попытки злоумышленников использовать его для анализа.

Управление маршрутизацией

Управление маршрутизацией означает выбор и непрерывное изменение различных доступных маршрутов между отправителем и приемником для того, чтобы препятствовать противнику в перехвате информации на определенном маршруте.

Доверенность

Доверенность означает выбор третьей стороны, с целью доверить ей контроль обменом между двумя объектами. Это может быть сделано, например, для того, чтобы предотвратить отказ от сообщения. Приемник может вовлечь третью сторону, которой можно доверить хранение запросов отправителя, и тем самым предотвратить последующее отрицание отправителем факта передачи сообщения.

Контроль доступа

Контроль доступа использует методы доказательства, что пользователь имеет право доступа к данным или ресурсам, принадлежащим системе. Примеры такого доказательства — пароли и PIN-коды.

Соотношение между услугами и механизмами

Таблица 1.2 показывает соотношение между услугами и механизмами информационной безопасности. Таблица приводит три механизма (шифрование, цифровая подпись и обмен сообщениями для опознавания), которые могут использоваться для того, чтобы обеспечить удостоверение подлинности. Таблица также показывает, что

шифрование может использоваться при трех услугах (конфиденциальности данных, целостности данных и аутентификации).

Таблица 1.2. Соотношение между услугами безопасности и механизмами обеспечения безопасности

Услуга безопасности	Механизм обеспечения безопасности
Конфиденциальность данных	Шифрование и управление маршрутизацией
Целостность данных	Шифрование, цифровая подпись, контрольные признаки целостности данных
Проверка полномочий	Шифрование, цифровая подпись, установление правомочности изменений
Исключение отказа от сообщений	Цифровая подпись, целостность данных и доверенность
Управление доступом	Механизм управления доступом

1.4. Методы

Механизмы, которые мы рассмотрели в предыдущих секциях, — это только теоретические рецепты. Для реализации информационной безопасности требуется небольшое число методов. Два из них наиболее распространены: один общий (криптография) и один специфический (стеганография).

Криптография

Некоторые механизмы информационной безопасности, перечисленные в предыдущей секции, могут быть реализованы с помощью криптографии. Криптография, слово с греческим происхождением, означает "тайна написанного". Однако мы используем этот термин, чтобы обозначить науку и искусство преобразования сообщений, которые делают их безопасными и придают иммунитет к атакам. Хотя в прошлом криптография заключалась только в шифровании и дешифровании сообщений с применением секретных ключей, сегодня

она определяется как совокупность трех различных механизмов: шифрование симметричными ключами, шифрование асимметричными ключами и хэширование. Ниже кратко рассмотрим эти три механизма.

Шифрование симметричными ключами

Шифрование симметричными ключами иногда называют шифрованием с секретным ключом или криптографией с секретным ключом. Например, объект, назовем его Алиса, может передать сообщение другому объекту, который называется Боб, по опасному каналу, для того, чтобы ее противник, который называется Ева, не смог понять содержание сообщения, просто подслушав его по каналу. Алиса зашифровала сообщение, используя алгоритм шифрования; Боб расшифровывает сообщение, используя алгоритм расшифровки. В шифровании симметричными ключами применяется единственный ключ засекречивания и для шифрования, и для расшифровки. Шифрование/дешифрование можно представить как электронный замок. При шифровании симметричным ключом Алиса помещает сообщение в блок и закрывает блок, используя совместный ключ засекречивания; Боб отпирает замок другим экземпляром того же ключа и извлекает сообщение.

Шифрование асимметричными ключами

При шифровании асимметричными ключами (иногда называемом шифрованием с открытыми ключами или криптографией с открытыми ключами) мы имеем ту же самую ситуацию, что и при шифровании симметричными ключами, но с небольшой разницей. Во-первых, мы имеем два ключа вместо одного: из них один открытый ключ (public key), другой — индивидуальный или секретный (private key). Для того чтобы передать защищенное сообщение Бобу, Алиса сначала зашифровала сообщение, используя открытый ключ Боба. Чтобы расшифровывать сообщение, Боб использует свой собственный секретный ключ.

Хэширование

При хэшировании из сообщения переменной длины может быть создан дайджест фиксированной длины, обычно намного меньшего размера, чем исходное сообщение. Сообщение и дайджест нужно передать Бобу. Дайджест используется, чтобы обеспечить проверку целостности данных, которая обсуждалась раньше.

Стеганография

Хотя эта книга базируется на криптографии как методике реализации механизмов безопасности, но все же кратко рассмотрим другую методику, которая в прошлом использовалась для засекречивания связи. В настоящее время она — стеганография — снова восстанавливается. Это слово происходит от греческого названия и означает "закрытую запись", в отличие от криптографии, означающей "секретную запись". Криптография скрывает содержание сообщение путем шифрования. Стеганография скрывает само сообщение непосредственно, закрывая его чем-нибудь.

Исторические примеры использования

История полна фактов и мифов об использовании стеганографии. В Китае военные сообщения писались на кусках тончайшего шелка и закатывались в маленький шар, который глотал посыльный. В Риме и Греции сообщения вырезались на кусочках древесины, которые потом опускались в воск, чтобы закрыть запись. Также использовались невидимые чернила (такие как луковый сок или соли аммиака) для записи секретного сообщения между строками безобидного текста или в конце бумаги; секретное сообщение выступало, когда эта бумага нагревалась или обрабатывалась каким-то веществом.

Недавно были изобретены другие методы. В некоторые безобидные письма могли бы быть записаны сообщения поверх письма карандашом, след которого видим только тогда, когда текст помещен под яркий источник света под углом. Нулевые шифры использовались, чтобы скрыть секретное сообщение в безвредном сообщении. Например, секретное сообщение можно составить, если первая или вторая буква в каждом слове бесполезна, а только закрывает истинное сообщение.

Микроточки также применялись для этой цели. Секретные сообщения были сфотографированы и уменьшены до размера точки и вставлялись в простые сообщения или периодически вставлялись в конце предложения.

Современное использование

Сегодня любая форма данных, такая как текст, изображение, аудио- или видеоинформация, может быть переведена в цифровую форму, и во время преобразования в цифровую форму или обработки можно в общие данные вставить секретную двоичную информацию. Такая скрытая информация не обязательно используется для сохранения тайны. Она может также использоваться как пометка, чтобы защитить авторское право, предотвратить вмешательство или внести дополнительную информацию, комментирующую текст для некоего получателя.

Скрывающие тексты. Для незаметной передачи секретных данных может быть задействован обычный текст. Есть несколько путей. Один из них — вставка двоичных символов. Например, мы можем использовать пробел между словами. Чтобы представить двоичную цифру `0`, используется одиночный пробел — и два пробела, чтобы представить двоичную цифру `1`. Представленное ниже короткое сообщение скрывает двоичное `8` -битовое представление буквы `A (01000001)` в коде ASCII.

Это учебник по изучению криптографии, а не по стеганографии
□ □□□ □ □□□ □□
0 1 0 0 0 0 0 1

В сообщении, которое приведено выше, два пробела содержатся между словами "`учебник`" и "`по`" и между "`по`" и "`стеганография`". Конечно, усложненное программное обеспечение может вставить пробелы, которые различаются минимально, чтобы скрыть код от непосредственного визуального распознавания.

Другой, более эффективный метод использует словарь слов, организованных согласно их грамматическим значениям (частям речи).

Мы можем, например, иметь словарь, содержащий 2 местоимения, 16 глаголов, 32 существительных. За каждым представителем этого словаря закреплен код. Предположим, что первый бит двоичных данных может быть представлен местоимением, каждое из которых имеет код (например, Я — это 0, а "мы" — это 1). Следующие пять битов могут быть представлены существительным (подлежащим в предложении). В нашем примере можно обозначить код 10010 словом "шофер". Следующие четыре бита могут быть представлены глаголом, (в примере — словом "веду", которое представляет код 0001), и последние пять бит — другим существительным (дополнение). В нашем примере "машину" — означает код 001001. Тогда можно договориться использовать скрывающий текст, который всегда применяет предложения местоимение — существительное — глагол — существительное. Секретные двоичные данные могут быть разделены на куски на 16 битов. Например, секретное сообщение "Hi", которое в ASCII отображается 0 10010 0001 001001, могло быть засекречено следующим предложением:

```
Я шофер веду машину
0 10010 0001 001001
```

Это — очень тривиальный пример. Реальный подход использует более усложненный алгоритм и большее разнообразие применяемых слов для одного и того же кода.

Методы скрытия, использующие изображения. Данные могут быть скрыты другим цветным изображением. Переведенные в цифровую форму изображения состоят из пикселей (элемент картинки), в котором обычно каждый пиксель использует 24 бита (три байта). Каждый байт представляет один из первичных цветов (красный, зеленый или синий). Мы можем поэтому иметь 2^8 различных оттенков каждого цвета. В методе, названном LSB (Last Significant Bit), самый младший бит каждого байта установлен на нуль. От этого изображение становится немного светлее в некоторых областях, но это обычно не замечается. Теперь мы можем скрыть двоичные данные в изображении, сохраняя или изменяя самый младший бит. Если наша двоичная цифра — 0, мы сохраняем бит; если это — 1, мы изменяем бит на 1. Этим способом мы можем скрыть символ (восемь битов ASCII) в трех пикселях. Последний бит

последнего пикселя не учитывается. Например, следующие три пикселя могут представить латинскую букву M (4D16 или, в двоичной системе, 0100 1101):

01010010	10111100	01010100
01011111	10111101	01100101
01111110	01001011	00010100

Другие методы скрытия. Возможны также другие методы скрытия. Секретное сообщение, например, может быть закрыто аудио- (звук и музыка) и видеоинформацией. И аудио, и видео сегодня подвергаются сжатию. Секретные данные могут быть внесены в информацию в процессе или перед сжатием. Теперь мы прекращаем обсуждение этих методов и рекомендуем интересующимся более специализированную литературу по стеганографии.

1.5. Рекомендованное чтение

Для более детального ознакомления с предметами, о которых шла речь в этой главе, начинающим можно рекомендовать нижеследующие книги и сайты. Символы, заключенные в скобки, рассматриваются как ссылки к списку литературы в конце книги.

Книги

Несколько книг рассматривают цели безопасности, нападения и механизмы. Мы рекомендуем [Bis05] и [Sta06].

Сайты

Больше информации о темах, обсужденных в этой главе, дают следующие сайты:

- ссылка: http://www.faqs.org/rfcs/rfc2828.html - http://www.faqs.org/rfcs/rfc2828.html
- ссылка: http://www.cs.binghamton.edu/~steflik/cs455/rfc/x800.pdf -

http://www.cs.binghamton.edu/~steflik/cs455/rfc/x800.pdf

1.6. Итоги

- Для информационной безопасности были определены три главные цели: конфиденциальность, целостность и готовность.
- Конфиденциальности информации угрожают два типа атак: вмешательство — и наблюдение за трафиком и его анализ. Четыре типа атак могут угрожать целостности информации: модификация, имитация источника, повторная передача информации и отказ от сообщения. Атака "прекращение обслуживания запроса" угрожает готовности информации.
- Организации, которые занимаются передачей данных и созданием сетей, такие как ITU-T или Internet Forum, определили несколько служб безопасности, предназначенных для целей информационной безопасности и защиты от атак. В этой главе рассматривались пять общих служб безопасности: конфиденциальность данных, целостность данных, установление подлинности, исключение отказа от сообщений и управление доступом.
- ITU-T также рекомендует некоторые механизмы обеспечения безопасности. В главе рассмотрены восемь из этих механизмов: шифрование, целостность данных, цифровая подпись, установление правомочности изменений, заполнение трафика, управление маршрутизацией, доверенность и управление доступом.
- Есть два метода — криптография и стеганография, которые могут реализовать некоторые или все механизмы. Криптография или "тайное письмо" включает скремблирование сообщения или создание дайджеста сообщения. Стеганография, или "закрытая запись", означает, что сообщение скрывается и закрывается другой информацией.

1.7. Набор для практики

Обзорные вопросы

1. Определите три цели безопасности.
2. Укажите различие между пассивными и активными атаками на секретную информацию. Назовите некоторые пассивные атаки. Назовите некоторые активные атаки.
3. Перечислите и определите пять служб безопасности, рассмотренные в этой главе.
4. Определите восемь механизмов безопасности, рассмотренные в этой главе.
5. Укажите различие между шифрованием и стеганографией.

Упражнения

1. Какая служба(ы) безопасности гарантируется при использовании каждого из следующих методов пересылки по почте в почтовом отделении?
    - Обычная почта
    - Обычная почта с подтверждением доставки
    - Обычная почта с доставкой и подписью получателя
    - Заказное письмо
    - Почта с объявленной ценностью
    - Зарегистрированная корреспонденция
2. Определить тип атаки на секретную информацию в каждом из следующих случаев:
    - Студент проникает в офис профессора, чтобы получить копию теста, который будет проведен на следующий день.
    - Студент дает чек на получение денег на `10$`, чтобы купить уже подержанную книгу. Потом он узнает, что чек был оплачен на `100$`.
    - Студент передает сотни запросов в день, используя фальшивый обратный адрес телефона другого студента.
3. Какие механизм(ы) безопасности реализованы в каждом из следующих случаев?
    - Университет требует идентификатор студента и пароль, чтобы позволить студенту получить доступ в школьный сервер.
    - Университетский сервер разъединяет студента, если он получил доступ в систему более чем два часа назад.

- Профессор отказывается передать студентам их оценки электронной почтой, если они не соответствуют студенческой идентификации, заранее назначенной профессором.
- Банк требует подписи клиента для изъятия клиентом денег.

4. Какая методика (криптография или стеганография) используется в каждом из следующих случаев для защиты конфиденциальности?
   - Студент пишет ответы на билеты на маленьком листочке бумаги, бумага свертывается и вставляется в шариковую ручку, а ручка передается другому студенту.
   - Чтобы передать сообщение, шпион заменяет каждый символ в сообщении символом, который был согласован заранее как замена другого символа.
   - Компания использует специальные чернила на своих чеках, чтобы предотвратить подделки.
   - Аспирантка использует водяные знаки, чтобы защитить свою работу, которая вывешена на ее сайте.
5. Какой механизм(ы) безопасности реализуется, когда человек подписывает форму при заполнении заявления на кредитную карту?

Модульная арифметика

Эта лекция необходима, чтобы подготовить читателя к дальнейшему разговору о криптографии. Лекция имеет несколько целей: рассмотреть арифметику целых чисел, которая базируется на теории делимости и нахождении наибольшего общего делителя, обратить внимание на важность модульной арифметики (арифметики над вычетами по модулю n) и операций в ней, потому что они широко используются в криптографии.

Криптография базируется на некоторых специфических областях математики, включая теорию чисел, линейную алгебру и алгебраические структуры. В этой лекции мы обсуждаем только те темы в вышеупомянутых областях, которые необходимы для понимания содержания следующих нескольких лекций. Читатели, которые знакомы с этими темами, могут пропустить лекцию полностью или частично. Подобные теоретические лекции встречаются всюду в этом курсе, когда это необходимо. Доказательства теорем и алгоритмов пропущены и даны только в приложении. Заинтересованный читатель может найти их в приложении Q.

Доказательства теорем и алгоритмов, обсуждаемых в этой лекции, можно просмотреть в приложении Q.

2.1. Арифметика целых чисел

В арифметике целых чисел мы используем множество целых чисел и несколько операций. Вы знакомы с этим множеством и соответствующими операциями, но они рассмотрены здесь, чтобы объяснить потом основы действий со сравнениями по модулю m.

Множество целых чисел

Множество целых чисел, обозначенных Z, содержит все числа (без дробей) от минус бесконечности до плюс бесконечности (рис. 2.1).

$$Z = \{\ldots\ldots -2, -1, 0, 1, 2, \ldots\ldots\}$$

Рис. 2.1. Множество целых чисел

Бинарные операции

В криптографии нас интересует три бинарных операции в приложении к множеству целых чисел. Бинарные операции имеют два входа и один выход. Для целых чисел определены три общих бинарных операции — сложение, вычитание и умножение. Каждая из этих операций имеет два входа (a и b) и выход (c), как это показано на рис. 2.2. Два входа принимают числа из множества целых чисел; выход выводит результат операции — число из множества целых чисел.

Обращаем внимание, что деление не относится к этой категории операций, потому что мы скоро убедимся, что этой операции нужны два выхода вместо одного.

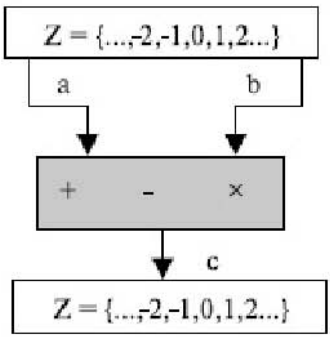


Рис. 2.2. Три бинарных операции для множества целых чисел

Пример 2.1

Следующие примеры показывают результаты трех бинарных операций на множестве двух целых чисел. Поскольку каждый вход может быть или положителен или отрицателен, мы имеем четыре случая для каждой операции.

Сложение	5+9=14	(-5)+9=4	5+(-9)=-4	(-5)+(-9)=-14
Вычитание	5-9=-4	(-5)-9=-14	5 - (-9)=14	(-5)- (-9)=+4
Умножение	5 x 9=45	(-5) x 9=-45	5 x (-9)=-45	(-5) x (-9)=45

Деление целых чисел

В арифметике целых чисел, если мы a делим на n, мы можем получить q и r. Отношения между этими четырьмя целыми числами можно показать как

$$a = q \times n + r$$

В этом равенстве a называется делимое ; q — частное ; n — делитель и r — остаток. Обратите внимание, что это — не операция, поскольку результат деления a на n — это два целых числа, q и r. Мы будем называть это уравнением деления.

Пример 2.2

Предположим, что a = 255, а n = 23. Мы можем найти q = 11 и r = 2, используя алгоритм деления, мы знаем из элементарной арифметики — оно определяется, как показано на рис. 2.3.

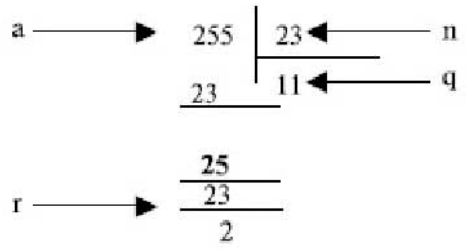


Рис. 2.3. Пример 2.2, нахождение частного и остатка

Большинство компьютерных языков может найти частное и остаток, используя заданные языком операторы. Например, на языке C оператор "/" может найти частное, а оператор "%" — остаток.

Два ограничения

Когда мы используем вышеупомянутое уравнение деления в криптографии, мы налагаем два ограничения. Первое требование: чтобы делитель был положительным целым числом (n > 0). Второе

требование: чтобы остаток был неотрицательным целым числом (r >= 0). Рисунок 2.4 показывает эти требования с двумя указанными ограничениями.

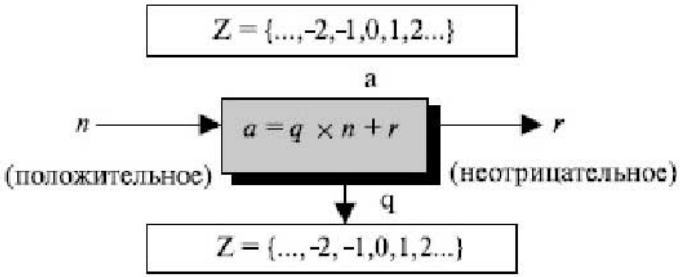


Рис. 2.4. Алгоритм деления целых чисел

Пример 2.3

Предположим, мы используем компьютер или калькулятор, а r и q отрицательны, при отрицательном a. Как можно сделать, чтобы выполнялось ограничение, что число r должно быть положительным? Решение простое: мы уменьшаем значение q на 1 и добавляем значение n к r, чтобы r стало положительным.

$$-255 = (-23 \times 11) + (-2) \leftrightarrow -255 = (-24 \times 11) + 9$$

Мы уменьшили (-23), получили (-24) и добавили 11 к (-2), чтобы получить + 9. Полученное равенство эквивалентно исходному.

Граф уравнения деления

Мы можем изобразить рассмотренные выше уравнения с двумя ограничениями на n и r на рис. 2.5 с помощью двух графов. Первый показывает случай, когда число a положительно; второй — когда отрицательно.

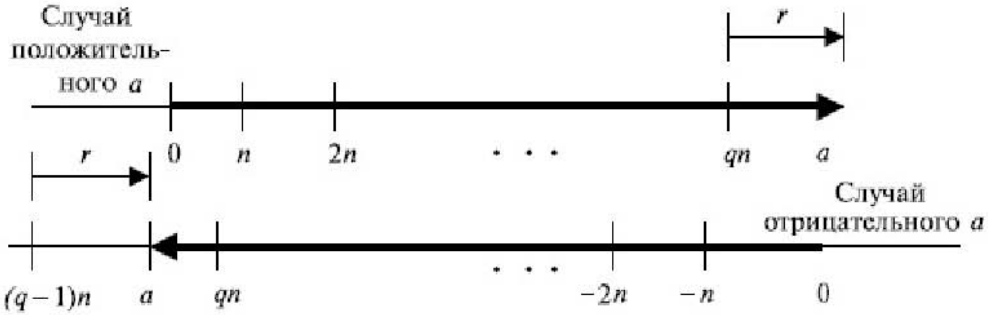


Рис. 2.5. Граф алгоритма деления

Граф начинается с нуля и показывает, как мы можем достигнуть точки, представляющей целое число a на линии. В случае положительного a мы должны перемещаться на величину $q \times n$ направо и затем добавить дополнительную величину r в том же самом направлении. В случае отрицательного a мы должны двигаться на величину $(q-1) \times n$ налево (число q в этом случае отрицательно) и затем дополнять число r в противоположном для указанного выше движения направлении. В обоих случаях значение r положительно.

Теория делимости

Теперь кратко обсудим теорию делимости — тема, с которой мы часто сталкиваемся в криптографии. Если a не равно нулю, а r = 0, в равенстве деления мы имеем

```
a = q x n
```

Мы тогда говорим, что a делится на n (или n — делитель a). Мы можем также сказать, что a делится без остатка на n. Когда мы не интересуемся значением q, мы можем записать вышеупомянутые отношения как n | a. Если остаток не является нулевым, то n не делит, и мы можем записать отношения как n † a.

Пример 2.4

a. Целое число 4 делит целое число 32, потому что $32 = 8 \times 4$. Это

можно отобразить как 4 | 32. Число 8 не делит число 42, потому что $42 = 5 \times 8 + 2$. В этом уравнении число 2 — остаток. Это можно отобразить как 8 † 42.

Пример 2.5

а. Отображение делимости 13 | 78, 7 | 98, -6 | 24, 4 | 44, и 11 | (-33).

б. Отображение неделимости 13 † 27, 7 † 50, – 6 † 23, 4 † 41, и 11 † (-32).

Свойства

Следующие несколько свойств теории делимости. Доказательства заинтересованный читатель может проверить в приложении Q.

Свойство 1: если a | 1, то a=±1.

Свойство 2: если a | b и b | a, то a=±b

Свойство 3: если a | b и b | c, то a | c

Свойство 4: если a | b и a | c, то a | (m x b + n x c), где m и n — произвольные целые числа.

Пример 2.6

а. Если 3 | 15 и 15 | 45 то, согласно третьему свойству, 3 | 45.

б. Если 3 | 15 и 3 | 9, то, согласно четвертому свойству, $3|(15 \times 2 + 9 \times 4)$, что означает 3 | 66.

Все делители

Положительное целое число может иметь больше чем один делитель. Например, целое число 32 имеет шесть делителей: 1, 2, 4, 8, 16 и 32.

Мы можем упомянуть два интересных свойства делителей положительных целых чисел.

Свойство 1: целое число 1 имеет только один делитель — само себя.

Свойство 2: любое положительное целое число имеет по крайней мере два делителя — 1 и само себя (но может иметь больше).

Наибольший общий делитель

Одно целое число, часто необходимое в криптографии, — наибольший общий делитель двух положительных целых чисел. Два положительных целых числа могут иметь много общих делителей, но только один наибольший общий делитель. Например, общие делители чисел 12 и 140 есть 1, 2 и 4. Однако наибольший общий делитель — 4 (см. рис. 2.6).

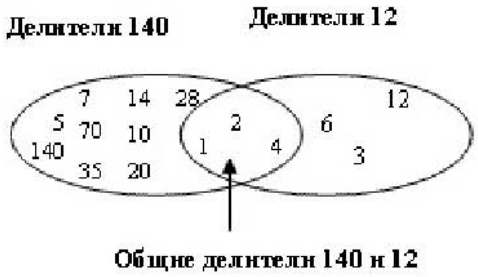


Рис. 2.6. Общие делители двух целых чисел

Наибольший общий делитель двух положительных целых чисел — наибольшее целое число, которое делит оба целых числа.

Алгоритм Евклида

Нахождение наибольшего общего делителя (НОД) двух положительных целых чисел путем составления списка всех общих делителей непригодно для достаточно больших чисел.

К счастью, больше чем 2000 лет назад математик по имени Эвклид

разработал алгоритм, который может найти наибольший общий делитель двух положительных целых чисел. Алгоритм Евклида основан на следующих двух фактах (доказательство см. в приложении Q):

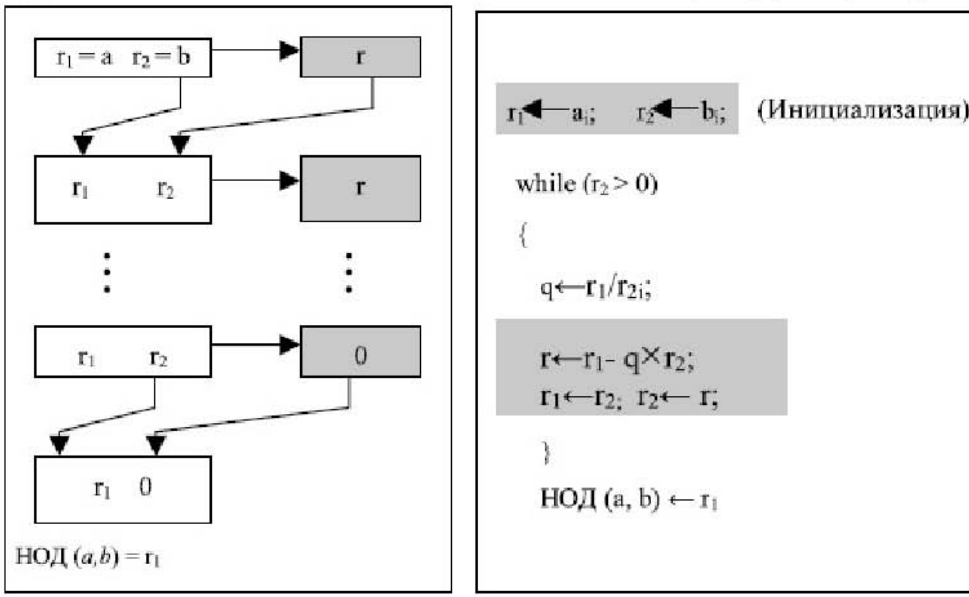
Факт 1: `НОД (a, 0) = a`

Факт 2: `НОД (a, b) = НОД (b, r)`, где `r` — остаток от деления `a` на `b`

Первый факт говорит, что если второе целое число — `0`, наибольший общий делитель равен первому числу. Второй факт позволяет нам изменять значение `a` на `b`, пока `b` не станет `0`. Например, вычисляя `НОД (36, 10)`, мы можем использовать второй факт несколько раз и один раз первый факт, как показано ниже.

НОД(36,10) = НОД(10,6) = НОД(6,4) = НОД(4,2) = НОД(2,0)

Другими словами, `НОД (36, 10) = 2`, `НОД (10, 6) = 2`, и так далее. Это означает, что вместо вычисления `НОД (36, 10)` мы можем найти `НОД (2, 0)`. Рисунок 2.7 показывает, как мы используем вышеупомянутые два факта, чтобы вычислить `НОД (a, b)`.



а) Процесс б) Алгоритм

Рис. 2.7. Алгоритм Евклида

Для определения НОД мы используем две переменные, r_1 и r_2, чтобы запоминать изменяющиеся значения в течение всего процесса. Они имеют начальное значение a и b. На каждом шаге мы вычисляем остаток от деления r_1 на r_2 и храним результат в виде переменной r. Потом заменяем r_1, на r_2 и r_2 на r и продолжаем шаги, пока r не станет равным 0. В этот момент процесс останавливается и НОД (a, b) равен r_1.

Пример 2.7

Нужно найти наибольший общий делитель 2740 и 1760.

Решение

Применим вышеупомянутую процедуру, используя таблицу. Мы присваиваем начальное значение r_1 2740 и r_2 значение 1760. В таблице также показаны значения q на каждом шаге. Мы имеем НОД (2740, 1760) = 20.

q	r_1	r_2	r
1	2740	1760	980
1	1760	980	780
1	980	780	200
3	780	200	180
1	200	180	20
9	180	20	0
	20	0	

Пример 2.8

Найти наибольший общий делитель `25` и `60`.

Решение

Мы выбрали этот конкретный пример, чтобы показать, что для алгоритма Евклида безразлично, если первое число меньше, чем второе. Все равно мы получаем правильный ответ `НОД (25, 60) = 5`.

q	r_1	r_2	R
0	25	60	25
2	60	25	10
2	25	10	5
2	10	5	0
	5	0	

Расширенный алгоритм Евклида

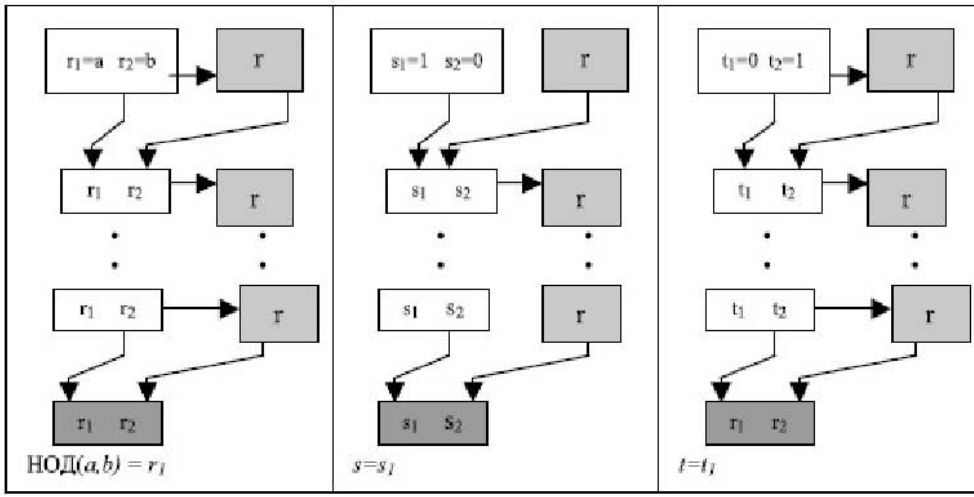
Даны два целых числа `a` и `b`. Нам зачастую надо найти другие два целых числа, `s` и `t`, такие, которые

$$s \times a + t \times b = НОД(a,b)$$

Расширенный алгоритм Евклида может вычислить `НОД (a, b)` и в

то же самое время вычислить значения `s` и `t`. Алгоритм и процесс такого вычисления показан на рис. 2.8.

Здесь расширенный алгоритм Евклида использует те же самые шаги, что и простой алгоритм Евклида. Однако в каждом шаге мы применяем три группы вычислений вместо одной. Алгоритм использует три набора переменных: `r`, `s` и `t`.



а) Процесс

$r_1 \leftarrow a; \quad r_2 \leftarrow b;$
$s_1 \leftarrow 1; \quad s_2 \leftarrow 0;$
$t_1 \leftarrow 0; \quad r_2 \leftarrow 1;$

(Инициализация)

while (r_2>0)
{
$q \leftarrow r_1 / r_2$;

$r \leftarrow r_1 - q \times r_2;$
$r_1 \leftarrow r_2; \; r_2 \leftarrow r;$

(обновление r)

$s \leftarrow s_1 - q \times s_2;$
$s_1 \leftarrow s_2; \; s_2 \leftarrow s;$

(обновление s)

$t \leftarrow t_1 - q \times t_2;$
$t_1 \leftarrow t_2; \; t_2 \leftarrow t;$

(обновление t)

}

НОД $(a, b) \leftarrow r_1; \; s \leftarrow s_1; \; t \leftarrow t_1;$

б) Алгоритм

Рис. 2.8. Расширенный алгоритм Евклида

На каждом шаге переменные r_1, r_2 и r используются так же, как в алгоритме Евклида. Переменным r_1 и r_2 присваиваются начальные значения a и b соответственно. Переменным s_1 и s_2 присваиваются начальные значения 1 и 0 соответственно. Переменным t_1 и t_2 присваиваются начальные значения 0 и 1, соответственно. Вычисления r, s и t одинаковы, но с одним отличием. Хотя r — остаток от деления r_1 на r_2, такого соответствия в других двух группах вычислений нет. Есть только одно частное, q, которое вычисляется как r_1/r_2 и используется для других двух вычислений.

Пример 2.9

Дано a = 161 и b = 28, надо найти НОД (a, b) и значения s и t.

Решение

$$r = r_1 - q \times r_2 \quad s = s_1 - qs_2 \quad t = t_1 - q \times t_2$$

Для отображения алгоритма мы используем следующую таблицу:

q	r_1	r_2	R	s_1	s_2	s	t_1	t_2	t
5	161	28	21	1	0	1	0	1	-5
1	28	21	7	0	1	-1	1	-5	6
3	21	7	0	1	-1	4	-5	6	-23
	7	0		-1	4		6	-23	

Мы получаем НОД (161, 28) = 7, s = –1 и t = 6. Ответы могут быть проверены, как это показано ниже.

$$(-1) \times 161 + 6 \times 28 = 7$$

Пример 2.10

Дано a = 17 и b = 0, найти НОД (a, b) и значения s и t.

Решение

Для отображения алгоритма мы используем таблицу.

q	r_1	r_2	R	s_1	s_2	s	t_1	t_2	t
	17	0		1	0		0	1	

Обратите внимание, что нам не надо вычислять `q`, `r` и `s`. Первое значение r_2 соответствует условию завершения алгоритма. Мы получаем `НОД (17, 0) = 17`, `s = 1` и `t = 0`. Это показывает, почему мы должны придавать начальные значения $s_1 = 1$ и $t_1 = 0$. Ответы могут быть проверены так, как это показано ниже:

(1 x 17) + (0 x 0) = 17

Пример 2.11

Даны `a = 0` и `b = 45`, найти `НОД (a, b)` и значения `s` и `t`.

Решение

Для отображения алгоритма мы используем следующую таблицу:

q	r_1	r_2	R	s_1	s_2	s	t_1	t_2	t
0	0	45	0	1	0	1	0	1	0
	45	0		0	1		0	1	

Мы получаем `НОД (0,45) = 45`, `s = 0` и `t = 1`. Отсюда ясно, что мы должны инициализировать s_2 равным `0`, а t_2 — равным `1`. Ответ может быть проверен, как это показано ниже:

(0 x 0) + (1 x 45) = 45

Линейные диофантовы уравнения

Хотя очень важное приложение расширенного алгоритма Евклида будет рассмотрено далее, здесь мы остановимся на другом приложении —

"нахождение решения линейных диофантовых уравнений двух переменных", а именно, уравнения $ax + by = c$. Мы должны найти значения целых чисел для x и y, которые удовлетворяют этому уравнению. Этот тип уравнения либо не имеет решений, либо имеет бесконечное число решений. Пусть $d = НОД\ (a, b)$. Если $d \nmid c$, то уравнение не имеет решения. Если $d | c$, то мы имеем бесконечное число решений. Одно из них называется частным, остальные — общими.

Линейное диофантово уравнение — это уравнение двух переменных: $ax + by = c$.

Частное решение

Если $d | c$, то можно найти частное решение вышеупомянутого уравнения, используя следующие шаги.

1. Преобразуем уравнение к $a_1x + b_1y = c_1$, разделив обе части уравнения на d. Это возможно, потому, что d делит a, b, и c в соответствии с предположением.
2. Найти s и t в равенстве $a_1s + b_1t = 1$, используя расширенный алгоритм Евклида.
3. Частное решение может быть найдено:

Частное решение: $x_0 = (c/d)s$ и $y_0 = (c/d)t$

Общие решения

После нахождения частного решения общие решения могут быть найдены:

Общие решения: $x = x_0 + k(b/d)$ и $y = y_0 - k(a/d)$, где k — целое число

Пример 2.12

Найти частные и общие решения уравнения 21x + 14y = 35.

Решение

Мы имеем d = НОД (21, 14) = 7. При 7|35 уравнение имеет бесконечное число решений. Мы можем разделить обе стороны уравнения на 7 и получим уравнение 3x + 2y = 5. Используя расширенный алгоритм Евклида, мы находим s и t, такие, что 3s + 2t = 1. Мы имеем S = 1 и t = −1. Решения будут следующие:

Частное решение : $x_0 = 5 \times 1 = 5$ и $y_0 = 5 \times (-1) = -5$ тогда 35/7 =5
Общие: x = 5+ k x 2 y= –5 – k x 3 где k — целое

Поэтому решения будут следующие (5, −5), (7, −8), (9, −11)...

Мы можем легко проверить, что каждое из этих решений удовлетворяет первоначальному уравнению.

Пример 2.13

Рассмотрим очень интересное приложение решения диофантовых уравнений в реальной жизни. Мы хотим найти различные комбинации объектов, имеющих различные значения. Например, мы хотим обменять денежный чек 100$ на некоторое число банкнот 20$ и несколько банкнот по 5$. Имеется много вариантов, которые мы можем найти, решая соответствующее диофантово уравнение 20x + 5y = 100. Обозначим d = НОД (20, 5) = 5 и 5|100. Уравнение имеет бесконечное число решений, но в этом случае приемлемы только несколько из них (только те ответы, в которых и x и y являются неотрицательными целыми числами). Мы делим обе части уравнения на 5, чтобы получить 4x + y = 20, и решаем уравнение 4s + t = 1. Мы можем найти s = 0 и t = 1, используя расширенный алгоритм Эвклида. Частное решение: $x_2 = 0 \times 20 = 0$ и $y_0 = 1 \times 20 = 20$. Общие решения с неотрицательными x и y — (0, 20), (1, 16), (2, 12), (3, 8), (4, 4), (5, 0). Остальная часть решений неприемлема, потому что y становится отрицательным. Кассир в банке должен спросить, какую из вышеупомянутых комбинаций мы хотим. Первое число в скобках

обозначает число банкнот по 20$; второе число обозначает число банкнот по 5$.

2.2. Модульная арифметика

Уравнение деления ($a = q \times n + r$), рассмотренное в предыдущей секции, имеет два входа (a и n) и два выхода (q и r). В модульной арифметике мы интересуемся только одним из выходов — остатком r. Мы не заботимся о частном q. Другими словами, когда мы делим a на n, мы интересуемся только тем, что значение остатка равно r. Это подразумевает, что мы можем представить изображение вышеупомянутого уравнения как бинарный оператор с двумя входами a и n и одним выходом r.

Операции по модулю

Вышеупомянутый бинарный оператор назван оператором по модулю и обозначается как mod. Второй вход (n) назван модулем. Вывод r назван вычетом. Рисунок 2.9 показывает отношение деления по сравнению с оператором по модулю.

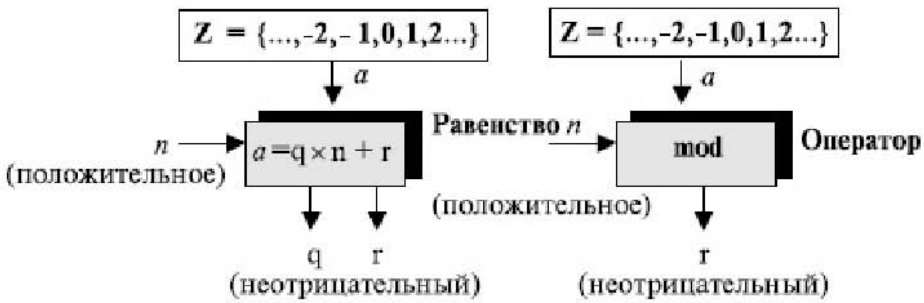


Рис. 2.9. Соотношение уравнения деления и оператора по модулю

Как показано на рис. 2.9, оператор по модулю (mod) выбирает целое число (a) из множества Z и положительный модуль (n). Оператор определяет неотрицательный остаток (r).

Мы можем сказать, что

`a mod n = r`

Пример 2.14

Найти результат следующих операций:

a. `27 mod 5`

b. `36 mod 12`

c. `-18 mod 14`

d. `-7 mod 10`

Решение

Мы ищем вычет `r`. Мы можем разделить `a` на `n` и найти `q` и `r`. Далее можно игнорировать `q` и сохранить `r`.

а. Разделим `27` на `5` - результат: `r = 2`. Это означает, что `27 mod 5 = 2`.

б. Разделим `36` на `12` — результат: `r = 0`. Это означает, что `36 mod 12 = 0`.

в. Разделим `(-18)` на `14` — результат: `r = -4`. Однако мы должны прибавить модуль `(14)`, чтобы сделать остаток неотрицательным. Мы имеем `r = -4 + 14 = 10`. Это означает, что `-18 mod 14 = 10`.

г. Разделим `(-7)` на `10` — результат: `r = -7`. После добавления модуля `-7` мы имеем `r = 3`. Это означает, что `-7 mod 10 = 3`.

Система вычетов: Zn

Результат операции по модулю `n` — всегда целое число между `0` и `n - 1`. Другими словами, результат `a mod n` — всегда неотрицательное целое число, меньшее, чем `n`. Мы можем сказать, что операция по модулю создает набор, который в модульной арифметике можно

понимать как систему наименьших вычетов по модулю n, или Z_n. Однако мы должны помнить, что хотя существует только одно множество целых чисел (Z), мы имеем бесконечное число множеств вычетов (Z_n), но лишь одно для каждого значения n. Рисунок 2.10 показывает множество Z_n и три множества Z_2, Z_6 и Z_{11}.

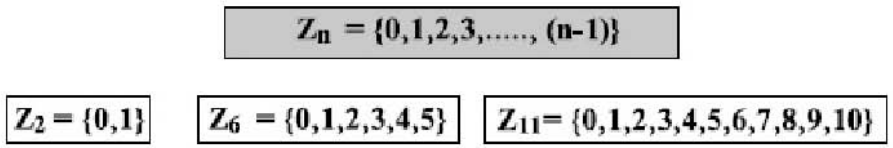


Рис. 2.10. Некоторые наборы Zn

Сравнения

В криптографии мы часто используем понятие сравнения вместо равенства. Отображение Z в Z_n не отображаются "один в один". Бесконечные элементы множества Z могут быть отображены одним элементом Z_n. Например, результат $2 \bmod 10 = 2, 12 \bmod 10 = 2, 22 \bmod 10 = 2$, и так далее. В модульной арифметике целые числа, подобные 2, 12, и 22, называются сравнимыми по модулю 10 $(\bmod 10)$. Для того чтобы указать, что два целых числа сравнимы, мы используем оператор сравнения ($\equiv$). Мы добавляем $\bmod n$ к правой стороне сравнения, чтобы определить значение модуля и сделать равенство правильным. Например, мы пишем:

$$2 \equiv 12 (mod\ 10) \quad 13 \equiv 23 (mod\ 10) \quad 34 \equiv 24 (mod\ 10) \quad -8 \equiv 12 (mod\ 10)$$
$$3 \equiv 8 (mod\ 5) \quad 8 \equiv 13 (mod\ 5) \quad 23 \equiv 33 (mod\ 5) \quad -8 \equiv 2 (mod\ 5)$$

Рисунок 2.11 показывает принцип сравнения. Мы должны объяснить несколько положений.

а. Оператор сравнения напоминает оператор равенства, но между ними есть различия. Первое: оператор равенства отображает элемент Z самого на себя; оператор сравнения отображает элемент Z на элемент Z_n. Второе: оператор равенства показывает, что наборы слева и справа

соответствуют друг другу "один в один", оператор сравнения — "многие — одному".

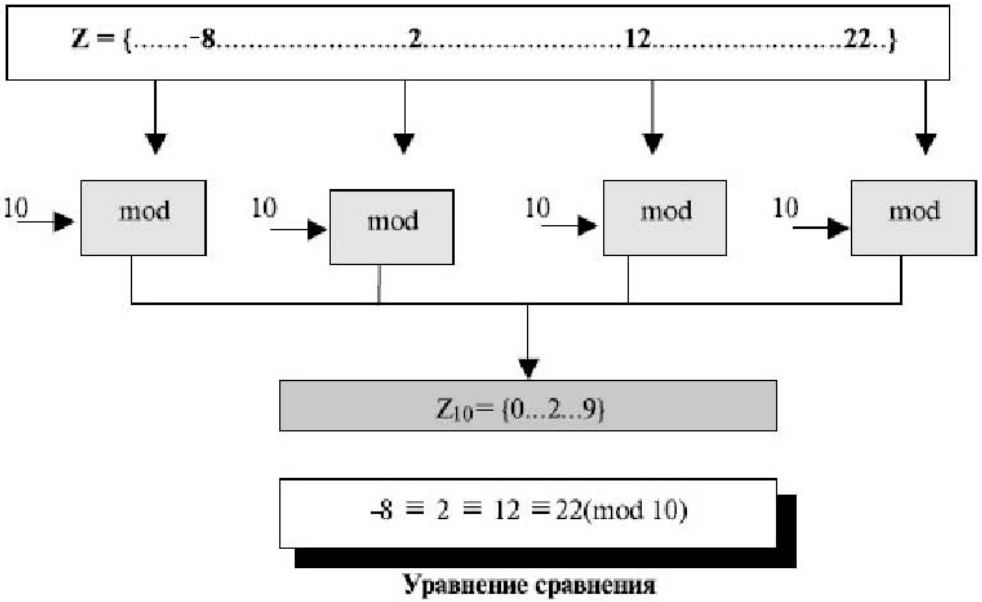


Рис. 2.11. Принцип сравнения

б. Обозначение (`mod n`), которое мы вставляем с правой стороны оператора сравнения, обозначает признак множества (Z_n). Мы должны добавить это обозначение, чтобы показать, какой модуль используется в отображении. Символ, используемый здесь, не имеет того же самого значения, как бинарный оператор в уравнении деления. Другими словами, символ `mod` в выражении `12 mod 10` — оператор; а сочетание (`mod 10`) в сравнении $2 \equiv 12 (\text{mod} 10)$ означает, что набор — Z_{10}.

Система вычетов

Система вычетов `[a]`, или $[a]_n$, — множество целых чисел, сравнимых по модулю `n`. Другими словами, это набор всех целых чисел, таких, что `x = a (mod n)`. Например, если `n = 5`, мы имеем

множество из пяти элементов [0], [1], [2], [3] и [4], таких как это показано ниже:

[0] = {…., –15, -10, –5, 0, 5, 10, 15, …}
[1] = {…., –14, –9, –4, 1, 6 , 11, 16,…}
[2] = {…., –13, –8, –3, 2, 7, 12, 17,…}
[3] = {...., –12, –7, –2, 3, 8, 13, 18,…}
[4] = {…., –11, –6, –1, 4, 9, 14, 19,…}

Целые числа в наборе [0] все дают остаток 0 при делении на 5 (сравнимы по модулю 5). Целые числа в наборе [1] все дают остаток 1 при делении на 5 (сравнимы по модулю 5), и так далее. В каждом наборе есть один элемент, называемый наименьшим (неотрицательным) вычетом. В наборе [0] это элемент 0 ; в наборе [1] — 1, и так далее. Набор, который показывает все наименьшие вычеты: $Z_5 = \{0, 1, 2, 3, 4\}$. Другими словами, набор Z_n — набор всех наименьших вычетов по модулю n.

Круговая система обозначений

Понятие "сравнение" может быть лучше раскрыто при использовании круга в качестве модели. Так же, как мы применяем линию, чтобы показать распределение целых чисел в Z, мы можем использовать круг, чтобы показать распределение целых чисел в Z_n.

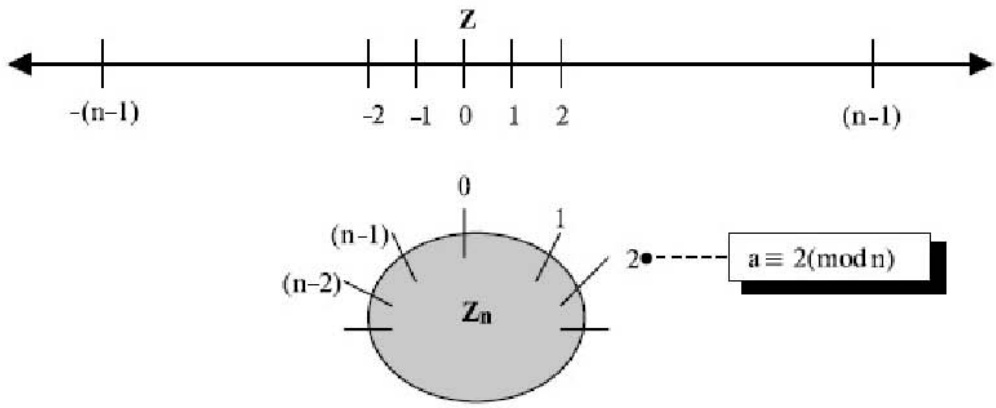


Рис. 2.12. Сравнение использования диаграмм для Z и Zn

Рисунок 2.12 позволяет сравнить два этих подхода. Целые числа от 0 до n–1 расположены равномерно вокруг круга. Все целые числа, сравнимые по модулю n, занимают одни и те же точки в круге. Положительные и отрицательные целые числа от Z отображаются в круге одним и тем же способом, соблюдая симметрию между ними.

Пример 2.15

Мы пользуемся сравнением по модулю в нашей ежедневной жизни; например, мы применяем часы, чтобы измерить время. Наша система часов использует арифметику по модулю 12. Однако вместо 0 мы берем отсечку 12, так что наша система часов начинается с 0 (или 12) и идет до 11. Поскольку наши сутки длятся 24 часа, мы считаем по кругу два раза и обозначаем первое вращение как утро до полудня, а второе — как вечер после полудня.

Операции в Zn

Три бинарных операции (сложение, вычитание и умножение), которые мы обсуждали для Z, могут также быть определены для набора Z_n. Результат, возможно, должен быть отображен в Z_n с использованием операции по модулю, как это показано на рис. 2.13.

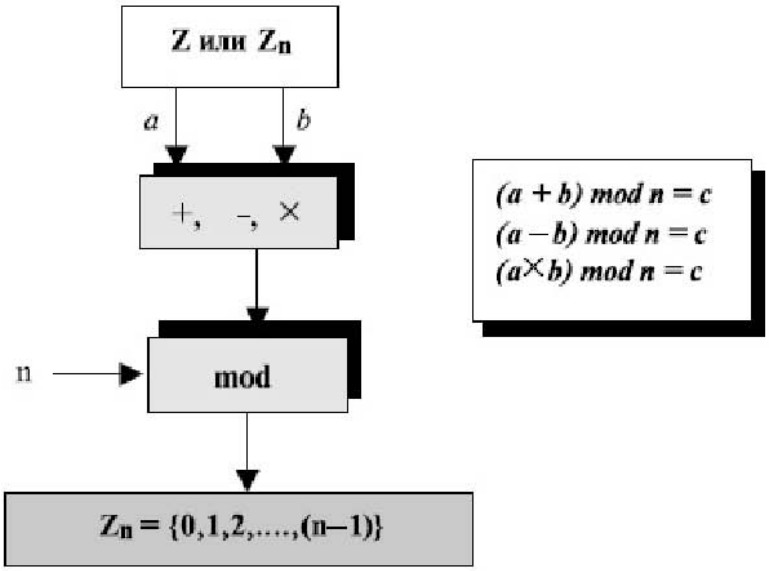


Рис. 2.13. Бинарные операции в Zn

Фактически применяются два набора операторов: первый набор — один из бинарных операторов $(+, -, \times)$; второй — операторы по модулю. Мы должны использовать круглые скобки, чтобы подчеркнуть порядок работ. Как показано на рис. 2.13, входы (a и b) могут быть членами Z или Z_n.

Пример 2.16

Выполните следующие операторы (поступающие от Z_n):

а. Сложение 7 и 14 в Z_{15}

б. Вычитание 11 из 7 в Z_{13}

в. Умножение 11 на 7 в Z_{20}

Решение

Ниже показаны два шага для каждой операции:

(14+7) mod 15 -> (21) mod 15 = 6
(7–11) mod 13 -> (-4) mod 13 = 9
(7x11) mod 20 -> (77) mod 20 = 17

Пример 2.17

Выполните следующие операции (поступающие от Z_n):

a. Сложение 17 и 27 в Z_{14}

b. Вычитание 43 из 12 в Z_{13}

c. Умножение 123 на -10 в Z_{19}

Решение

Ниже показаны два шага для каждой операции:

(17 + 27) mod 14 -> (44) mod 14 = 2
(12 – 43) mod 13 -> (–31) mod 13 = 8
((123) x (–10)) mod 19 -> (–1230) mod 19 = 5

Свойства

Мы уже упоминали, что два входа для трех бинарных операторов в сравнении по модулю могут использовать данные из Z или Z_n. Следующие свойства позволяют нам сначала отображать два входа к Z_n (если они прибывают от Z) перед выполнением этих трех бинарных операторов $(+, -, \times)$. Заинтересованные читатели могут найти доказательства для этих свойств в приложении Q.

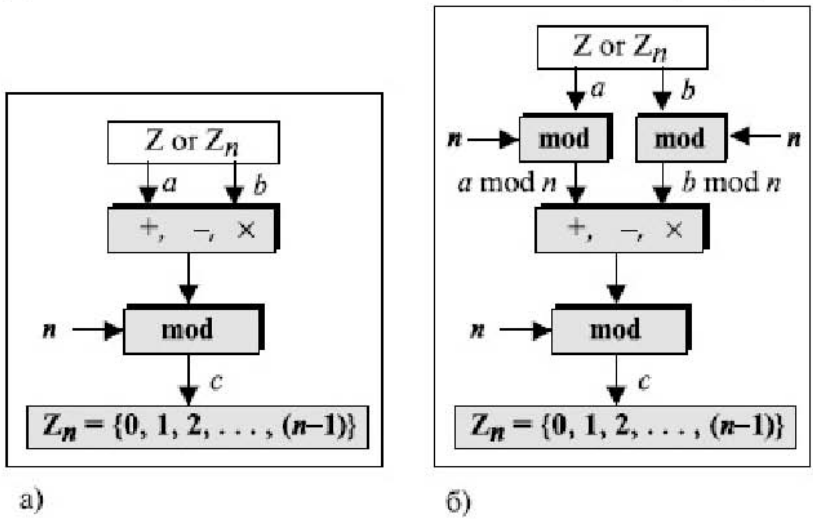


Рис. 2.14. Свойства оператора mod

Первое свойство: `(a + b) mod n = [(a mod n) + (b mod n)] mod n`

Второе свойство: `(a – b) mod n = [(a mod n) - (b mod n)] mod n`

Третье свойство: `(a x b) mod n = [(a mod n) x (b mod n)] mod n`

Рисунок 2.14 показывает процесс до и после применения указанных выше свойств. Хотя по рисунку видно, что процесс с применением этих свойств более длинен, мы должны помнить, что в криптографии мы имеем дело с очень большими целыми числами. Например, если мы умножаем очень большое целое число на другое очень большое целое число, которое настолько большое, что не может быть записано в компьютере, то применение вышеупомянутых свойств позволяет уменьшить первые два операнда прежде, чем начать умножение. Другими словами, перечисленные свойства позволяют нам работать с меньшими числами. Этот факт станет понятнее при обсуждении экспоненциальных операций в последующих лекциях.

Пример 2.18

Следующие примеры показывают приложение вышеупомянутых свойств.

1. $(1723345 + 2124945) \bmod 11 = (8 + 9) \bmod 11 = 6$
2. $(1723345 - 2124945) \bmod 11 = (8 - 9) \bmod 11 = 10$
3. $(1723345 \times 2124945) \bmod 11 = (8 \times 9) \bmod 11 = 6$

Пример 2.19

В арифметике мы часто должны находить остаток от степеней числа 10 при делении на целое число. Например, мы должны найти $10 \bmod 3$, $10^2 \bmod 3$, $10^3 \bmod 3$, и так далее. Мы также должны найти $10 \bmod 7$, $10^2 \bmod 7$, $10^3 \bmod 7$, и так далее. Третье свойство модульных операторов, упомянутое выше, делает жизнь намного проще.

$10^n \bmod x = (10 \bmod x)^n$ Применение третьего свойства n раз.

Мы имеем

$10 \bmod 3 = 1 \rightarrow 10^n \bmod 3 = (10 \bmod 3)^n = 1$
$10 \bmod 9 = 1 \rightarrow 10^n \bmod 9 = (10 \bmod 9)^n = 1$
$10 \bmod 7 = 3 \rightarrow 10^n \bmod 7 = (10 \bmod 7)^n = 3^n \bmod 7$

Пример 2.20

Мы уже говорили, что в арифметике остаток от целого числа, разделенного на 3, такой же, как остаток от деления суммы его десятичных цифр. Другими словами, остаток от деления 6371 равен остатку от деления суммы его цифр (17), на 3. Мы можем доказать, что это утверждение использует свойства модульного оператора. Запишем целое число как сумму его цифр, умноженных на степени 10.

$a = a_n 10^n + \ldots\ldots\ldots + a_1 10^1 + a_0 10^0$
Например: $6371 = 6 \times 10^3 + 3 \times 10^2 + 7 \times 10^1 + 1 \times 10^0$

Теперь мы можем применить модульную операцию к двум сторонам равенства и использовать результат предыдущего примера, где остаток

$10^n \bmod 3$ равен 1.

$$a \bmod 3 = (a_n \times 10^n + \ldots + a_1 \times 10^1 + a_0 \times 10^0) \bmod 3$$
$$= (a_n \times 10^n) \bmod 3 + \ldots + (a_1 \times 10^1) \bmod 3 + (a_0 \times 10^0 \bmod 3) \bmod 3$$
$$= (a_n \bmod 3) \times (10^n \bmod 3) + \ldots + (a_1 \bmod 3) \times (10^1 \bmod 3) +$$
$$(a_0 \bmod 3) \times (10^0 \bmod 3) \bmod 3$$
$$= ((a_n \bmod 3) + \ldots + (a_1 \bmod 3) + (a_0 \bmod 3)) \bmod 3$$
$$= (a_n + \ldots + a_1 + a_0) \bmod 3$$

Инверсии

Когда мы работаем в модульной арифметике, нам часто нужно найти операцию, которая позволяет вычислить величину, обратную заданному числу. Мы обычно ищем аддитивную инверсию (оператор, обратный сложению) или мультипликативную инверсию (оператор, обратный умножению).

Аддитивная инверсия

В Z_n два числа a и b аддитивно инверсны друг другу, если $b = n - a$. Например,

$$a + b \equiv 0 (mod\ n)$$

В Z_n аддитивная инверсия числу a может быть вычислена как $b = n - a$. Например, аддитивная инверсия 4 в Z_{10} равна $10 - 4 = 6$.

В модульной арифметике каждое целое число имеет аддитивную инверсию. Сумма целого числа и его аддитивной инверсии сравнима с 0 по модулю n .

Обратите внимание, что в модульной арифметике каждое число имеет аддитивную инверсию, и эта инверсия уникальна; каждое число имеет одну и только одну аддитивную инверсию. Однако инверсия числа может быть непосредственно тем же самым числом.

Пример 2.21

Найдите все взаимно обратные пары по сложению в Z_{10}.

Решение

Даны шесть пар аддитивных инверсий — (0, 0), (1, 9), (2, 8), (3, 7), (4, 6) и (5, 5). В этом списке 0 — инверсия самому себе; так же и 5. Обратите внимание: аддитивные инверсии обратны друг другу; если 4 — аддитивная инверсия 6, тогда 6 — также аддитивная инверсия числу 4.

Мультипликативная инверсия

В Z_n два числа a и b мультипликативно инверсны друг другу, если

$$a \times b \equiv 1 (mod\ n)$$

Например, если модуль равен 10, то мультипликативная инверсия 3 есть 7. Другими словами, мы имеем $(3 \times 7) \bmod 10 \equiv 1$.

В модульной арифметике целое число может или не может иметь мультипликативную инверсию. Целое число и его мультипликативная инверсия сравнимы с 1 по модулю n .

Может быть доказано, что a имеет мультипликативную инверсию в Z_n, если только НОД(n, a) = 1. В этом случае говорят, что a и n взаимно простые.

Пример 2.22

Найти мультипликативную инверсию 8 в Z_{10}.

Решение

Мультипликативная инверсия не существует, потому что $HOD\,(10, 8) = 2 \neq 1$. Другими словами, мы не можем найти число

между 0 и 9, такое, что при умножении на 8 результат сравним с 1 по mod 10.

Пример 2.23

Найти все мультипликативные инверсии в Z_{10}.

Решение

Есть только три пары, удовлетворяющие условиям существования мультипликативной инверсии: (1, 1), (3, 7) и (9, 9). Числа 0, 2, 4, 5, 6 и 8 не имеют мультипликативной инверсии.

Мы можем проверить, что

(1 x 1) mod 10 = 1 (3 x 7) mod 10 = 1 (9 x 9) mod 10 = 1

Пример 2.24

Найти все мультипликативные обратные пары в Z_{11}.

Решение

Мы имеем следующие пары: (1, 1), (2, 6), (3, 4), (5, 9), (7, 8) и (10, 10). При переходе от Z_{10} к Z_{11} число пар увеличивается. При Z_{11} НОД (11, a) = 1 (взаимно простые) для всех значений a, кроме 0. Это означает, что все целые числа от 1 до 10 имеют мультипликативные инверсии.

Целое число a в Z_n имеет мультипликативную инверсию тогда и только тогда, если НОД (n, a) = 1(mod n)

Расширенный алгоритм Евклида, который мы обсуждали ранее в этой лекции, может найти мультипликативную инверсию b в Z_n, когда даны n и b и инверсия существует. Для этого нам надо заменить первое целое число a на n (модуль). Далее мы можем утверждать, что алгоритм может найти s и t, такие, что $s \times n + b \times t = HOD(n, b)$. Однако если мультипликативная инверсия b существует, НОД (n, b) должен

быть 1. Так что уравнение будет иметь вид

(s x n) + (b x t) = 1

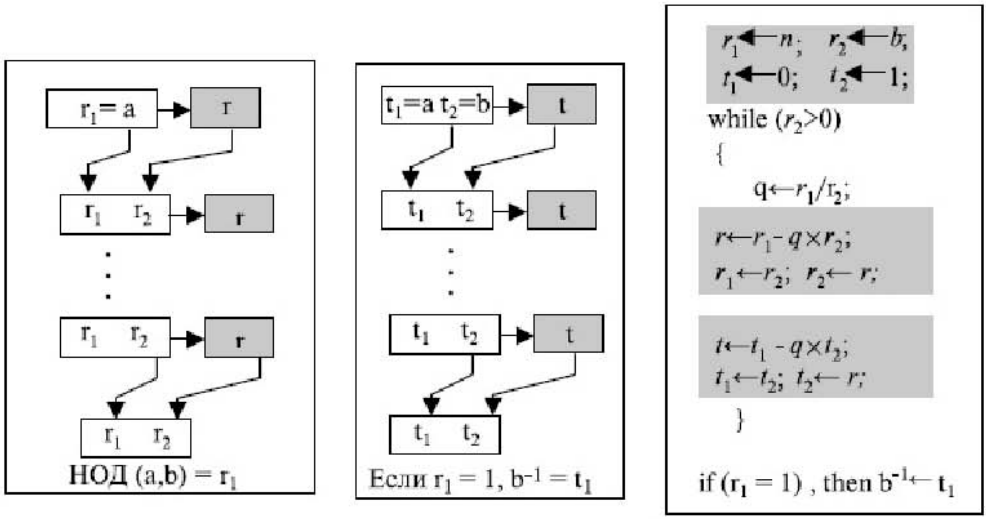
Теперь мы применяем операции по модулю к обеим сторонам уравнения. Другими словами, мы отображаем каждую сторону к Z_n. Тогда мы будем иметь

(s x n + b x t) mod n =1 mod n
[(s x n) mod n] + [(b x t) mod n] = 1 mod n
0 + [(b x t) mod n] = 1
(b x t) mod n =1 -> Это означает, что t – это мультипликативная инверс

Обратите внимание, что $[(s \times n) \bmod n]$ на третьей строке — 0, потому что, если мы делим $(s \times n)n$, частное — s, а остаток — 0.

Расширенный алгоритм Евклида находит мультипликативные инверсии b в Z_n, когда даны n и b и НОД (n, b) = 1. Мультипликативная инверсия b — это значение t, отображенное в Z_n.

Рисунок 2.15 показывает, как мы находим мультипликативную инверсию числа, используя расширенный алгоритм Евклида.



а) Процесс б) Алгоритм

Рис. 2.15. Применение расширенного алгоритма Евклида для поиска мультипликативной инверсии

Пример 2.25

Найти мультипликативную инверсию 11 в Z_{26}.

Решение

Мы используем таблицу, аналогичную одной из тех, которые мы уже применяли прежде при данных $r_1 = 26$ и $r_2 = 11$. Нас интересует только значение t.

q	r_1	r_2	r	t_1	t_2	t
2	26	11	4	0	1	-2
2	11	4	3	1	-2	5
1	4	3	1	-2	5	-7
3	3	1	0	5	-7	26
	1	0		-7	26	

НОД (26, 11) = 1, что означает, что мультипликативная инверсия 11 существует. Расширенный алгоритм Евклида дает $t_1 = (-7)$.

Мультипликативная инверсия равна (−7) mod 26 = 19. Другими словами, 11 и 19 — мультипликативная инверсия в Z_{26}. Мы можем видеть, что $(11 \times 19) \bmod 26 = 209 \bmod 26 = 1$.

Пример 2.26

Найти мультипликативную инверсию 23 в Z_{100}.

Решение

Мы используем таблицу, подобную той, которую применяли до этого при $r_1 = 100$ и $r_2 = 23$. Нас интересует только значение t.

q	r_1	r_2	r	t_1	t_2	t

4	100	23	8	0	1	-4
2	23	8	7	1	-4	19
1	8	7	1	-4	9	-13
7	7	1	0	9	-13	100
	1	0		-13	100	

НОД (100, 23) = 1, что означает, что инверсия 23 существует. Расширенный Евклидов алгоритм дает $t_1 = -13$. Инверсия — (−13) mod 100 = 87. Другими словами, 13 и 87 — мультипликативные инверсии в Z_{100}. Мы можем видеть, что

$$(23 \times 87) \bmod 100 = 2001 \bmod 100 = 1.$$

Пример 2.27

Найти инверсию 12 в Z_{26}.

Решение

Мы используем таблицу, подобную той, которую мы применяли раньше при $r_1 = 26$ и $r_2 = 12$.

q	r_1	r_2	r	t_1	t_2	t
2	26	12	2	0	1	
6	12	2	0	1	-2	
	2	0		-2	13	

$HOD\,(26, 12) = 2 \neq 1$, что означает отсутвствие для числа 12 мультипликативной инверсии в Z_{26}

Сложение и умножение таблиц

Рисунок 2.16 показывает две таблицы для сложения и умножения. При сложении таблиц каждое целое число имеет аддитивную инверсию. Обратные пары могут быть найдены, если результат их сложения —

ноль. Мы имеем (0, 0), (1, 9), (2, 8), (3, 7), (4, 6) и (5, 5). При умножении таблиц мы получаем только три мультипликативных пары (1, 1), (3, 7) и (9, 9). Пары могут быть найдены, когда результат умножения равен 1. Обе таблицы симметричны по диагонали, от левой вершины к нижней вершине справа. При этом можно обнаружить свойства коммутативности для сложения и умножения (a+b = b+a и $a \times b = b \times a$). Таблица сложения также показывает, что каждый ряд или колонка может поменяться с другим рядом или колонкой. Для таблицы умножения это неверно.

	0	1	2	3	4	5	6	7	8	9
0	0	1	2	3	4	5	6	7	8	9
1	1	2	3	4	5	6	7	8	9	0
2	2	3	4	5	6	7	8	9	0	1
3	3	4	5	6	7	8	9	0	1	2
4	4	5	6	7	8	9	0	1	2	3
5	5	6	7	8	9	0	1	2	3	4
6	6	7	8	9	0	1	2	3	4	5
7	7	8	9	0	1	2	3	4	5	6
8	8	9	0	1	2	3	4	5	6	7
9	9	0	1	2	3	4	5	6	7	8

Таблица сложения в Z_{10}

	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7	8	9
2	0	2	4	6	8	0	2	4	6	8
3	0	3	6	9	2	5	8	1	4	7
4	0	4	8	2	6	0	4	8	2	6
5	0	5	0	5	0	5	0	5	0	5
6	0	6	2	8	4	0	6	2	8	4
7	0	7	4	1	8	5	2	9	6	3
8	0	8	6	4	2	0	8	6	4	2
9	0	9	8	7	6	5	4	3	2	1

Таблица умножения в Z_{10}

Рис. 2.16. Таблицы сложения и умножения для Z10

Различные множества для сложения и умножения

В криптографии мы часто работаем с инверсиями. Если отправитель посылает целое число (например, ключ для шифрования слова), приемник применяет инверсию этого целого числа (например, ключ декодирования). Если это действие (алгоритм шифрования/декодирования) является сложением, множество Z_n может быть использовано как множество возможных ключей, потому что каждое целое число в этом множестве имеет аддитивную инверсию. С другой стороны, если действие (алгоритм шифрования/декодирования) — умножение, Z_n не может быть множеством возможных ключей, потому что только некоторые члены этого множества имеют

мультипликативную инверсию. Нам нужно другое множество, которое является подмножеством Z_n и включает в себя только целые числа, и при этом в Z_n они имеют уникальную мультипликативную инверсию. Это множество обозначается Z_{n*}. Рисунок 2.17 показывает некоторые случаи двух множеств. Обратите внимание, что множество Z_{n*} может быть получено из таблицы умножения типа показанной на рис. 2.16.

Каждый член Z_n имеет аддитивную инверсию, но только некоторые члены имеют мультипликативную инверсию. Каждый член Z_{n*} имеет мультипликативную инверсию, но только некоторые члены множества имеют аддитивную инверсию.

Мы должны использовать Z_n , когда необходимы аддитивные инверсии; мы должны использовать Z_{n*} , когда необходимы мультипликативные инверсии.

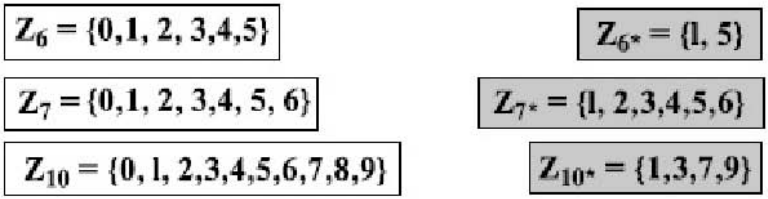


Рис. 2.17. Некоторые множества Zn и Zn*

Еще два множества

Криптография часто использует еще два множества: Z_p, и Z_{p*}. Модули в этих двух множествах — простые числа. Простые числа будут обсуждаться в следующих лекциях; пока можно сказать, что простое число имеет только два делителя: целое число 1 и само себя.

Множество Z_p — то же самое, что и Z_n, за исключением того, что n — простое число. Z_p содержит все целые числа от 0 до $p - 1$. Каждый элемент в Z_p имеет аддитивную инверсию; каждый элемент кроме 0 имеет мультипликативную инверсию.

Множество Z_{p*} — то же самое, что Z_{n*}, за исключением того, что Z_{p*} содержит все целые числа от 1 до $p - 1$. Каждый элемент в Z_p имеет аддитивную и мультипликативную инверсии. Z_{p*} очень хороший кандидат, когда мы нуждаемся во множестве, которое поддерживает аддитивную и мультипликативную инверсии.

Ниже показаны два множества, когда $p = 13$.

$Z_{13} = \{0,1,2,3,4,5,6,7,8,9,10,11,12\}$,
$Z_{13*} = \{1,2,3,4,5,6,7,8,9,10,11,12\}$,

Сравнения и матрицы

В данной лекции рассматриваются матрицы и операции с матрицами вычетов, которые широко используются в криптографии. Используя матрицы вычетов решается набор уравнений сравнения.

3.1. Матрицы

В криптографии мы должны обрабатывать матрицы. Хотя эта тема принадлежит специальному разделу алгебры, который называется линейной алгеброй, необходим краткий обзор матриц для подготовки к изучению криптографии. Читатели, знакомые с этими вопросами, могут пропустить часть или весь этот раздел. Раздел начинается с некоторых определений и примеров использования матрицы в модульной арифметике.

Определения

Матрица — прямоугольный массив, содержащий `l x m` элементов, в которых `l` — число строк, `m` — число столбцов. Матрица обычно обозначается заглавной буквой, такой, как `A`. Элемент a_{ij} расположен в `i` -той строке и `j` -том столбце. Хотя элементы матрицы могут быть любым множеством чисел, мы обсуждаем только матрицы с элементами в `Z`. Пример матрицы с `m` столбцами и `l` строками

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{22} & \dots & a_{2m} \\ \dots & & & \\ a_{l1} & a_{l2} & \dots & a_{lm} \end{pmatrix}$$

Если матрица имеет только одну строку (`l = 1`), она называется матрицей-строкой ; если она имеет только один столбец (`m = 1`), то называется матрицей-столбцом. Матрица называется квадратной, если число строк равно числу столбцов (`l = m`) и содержит элементы a_{11}, a_{22}, ……, a_{mm}. Матрица обозначается `0`, если все строки и все столбцы

содержат нули. Единичная матрица обозначается I, если она квадратная и содержит все единицы на главной диагонали и все нули на других местах. Рисунок 3.2 показывает некоторые примеры матриц с элементами из Z.

Рис. 3.2. Примеры матриц

Операции и уравнения

В линейной алгебре для матриц определены одно уравнение (равенство) и четыре операции (сложение, вычитание, умножение и скалярное умножение).

Равенство

Две матрицы равны, если они имеют одинаковое число строк и столбцов и соответствующие элементы равны. Другими словами, A = B, если мы имеем $a_{ij} = b_{ij}$ для всех i и j.

Сложение и вычитание

Операция сложения двух матриц может применяться, если матрицы имеют одинаковое число столбцов и строк. Сложение записывают как C =A + B. В этом случае полученная в результате матрица C имеет тот же самый номер строк и столбцов, как A или B. Каждый элемент C — сумма двух соответствующих элементов A и B: $a_{ij} + b_{ij}$.

Операция вычитания производится аналогично сложению, за исключением того, что каждый элемент B вычитается из соответствующего элемента A: $d_{ij} = a_{ij} - b_{ij}$.

Пример 3.1

Ниже показан пример сложения и вычитания.

$$\begin{pmatrix} 12 & 4 & 4 \\ 11 & 12 & 30 \end{pmatrix} = \begin{pmatrix} 5 & 2 & 1 \\ 3 & 2 & 10 \end{pmatrix} + \begin{pmatrix} 7 & 2 & 3 \\ 8 & 10 & 20 \end{pmatrix}$$

$$C = A + B$$

$$\begin{pmatrix} -2 & 0 & -2 \\ -5 & -8 & -10 \end{pmatrix} = \begin{pmatrix} 5 & 2 & 1 \\ 3 & 2 & 10 \end{pmatrix} - \begin{pmatrix} 7 & 2 & 3 \\ 8 & 10 & 20 \end{pmatrix}$$

$$C = A - B$$

Умножение

Две матрицы различного размера могут быть перемножены, если число столбцов первой матрицы совпадает с числом строк второй матрицы. Если A — матрица размера l x m, а матрица B размера m x p, то произведением будет матрица C размером l x p. Если элемент матрицы A обозначить a_{ij}, а каждый элемент матрицы B обозначить b_{jk}, то элемент матрицы C — c_{ik} — вычисляется следующим образом:

$$c_{ik} = \Sigma a_{ij} x b_{jk} = a_{i1} 1_j + a_{i2} x b_{2j} + \ldots + a_{im} x b_{mj}$$

Пример 3.2

Рисунок 3.3 показывает произведение матрицы-строки (1×3) на матрицу-столбец (3×1). В результате получаем матрицу размером 1×1.

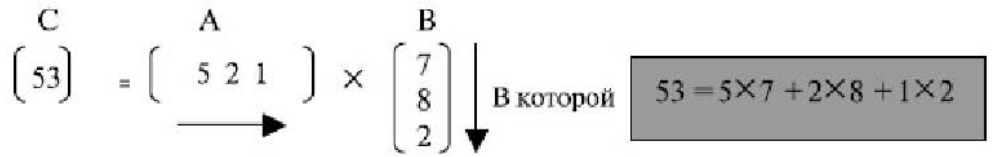


Рис. 3.3. Умножение матрицы-строки на матрицу-столбец

Пример 3.3

Рисунок 3.4 показывает произведение матрицы 2×3 на матрицу 3×4. В результате получаем матрицу 2×4

$$\underset{\mathbf{C}}{\begin{pmatrix} 52 & 18 & 14 & 9 \\ 41 & 21 & 22 & 7 \end{pmatrix}} = \underset{\mathbf{A}}{\begin{pmatrix} 5 & 2 & 1 \\ 3 & 2 & 4 \end{pmatrix}} \times \underset{\mathbf{B}}{\begin{pmatrix} 7 & 3 & 2 & 1 \\ 8 & 0 & 0 & 2 \\ 1 & 3 & 4 & 0 \end{pmatrix}}$$

Рис. 3.4. Умножение матрицы 2 x 3 на матрицу 3 x 4.

Скалярное умножение

Мы можем также умножить матрицу на число (называемое скаляр). Если `A` — матрица $l \times m$ и `x` — скаляр, то `C = xA` — матрица $l \times m$, в которой $c_{ij} = x \times a_{ij}$.

$$\underset{\mathbf{B}}{\begin{pmatrix} 15 & 6 & 3 \\ 9 & 6 & 12 \end{pmatrix}} = 3 \times \underset{\mathbf{A}}{\begin{pmatrix} 5 & 2 & 1 \\ 3 & 2 & 4 \end{pmatrix}}$$

Рис. 3.5. Скалярное умножение

Пример 3.4

Рисунок 3.5 показывает пример скалярного умножения.

Детерминант

Детерминант — квадратная матрица `A` размера $m \times m$, обозначаемая как `det (A)` — скалярное вычисление рекурсивно, как это показано ниже:

1. $If\ m = 1,\ \det(A) = a_{11}$

2. $If\ m > 1,\ \det(A) = \sum_{i=1\ldots m} (-1)^{i+j} \times a_{ij} \times \det(A_{ij})$

где A_{ij} получается из A удалением i -той строки j -того столбца.

Детерминант определяется только для квадратной матрицы.

Пример 3.5

Рисунок 3.6 показывает, как можно вычислить детерминант матрицы 2×2, базируясь на детерминанте матрицы 1×1 и используя приведенное выше рекурсивное определение. Пример доказывает, что когда m 1 или 2, это позволяет найти детерминант матрицы достаточно просто.

$$\det\begin{bmatrix}5 & 2\\ 3 & 4\end{bmatrix} = (-1)^{1+1} \times 5 \times \det[4] + (-1)^{1+2} \times 2 \times \det[3] \longrightarrow 5 \times 4 - 2 \times 3 = 14$$

или

$$\det\begin{bmatrix}a_{11} & a_{12}\\ a_{21} & a_{22}\end{bmatrix} = a_{11} \times a_{22} - a_{12} \times a_{21}$$

Рис. 3.6. Вычисление детерминанта матрицы 2 x 2

Пример 3.6

Рисунок 3.7 показывает вычисление детерминанта матрицы 3×3.

$$\det\begin{bmatrix}5 & 2 & 1\\ 3 & 0 & -4\\ 2 & 1 & 6\end{bmatrix} = (-1)^{1+1} \times 5 \times \det\begin{bmatrix}0 & -4\\ 1 & 6\end{bmatrix} + (-1)^{1+2} \times 2 \times \det\begin{bmatrix}3 & -4\\ 2 & 6\end{bmatrix} +$$

$$+(-1)^{1+3} \times 1 \times \det\begin{bmatrix}3 & 0\\ 2 & 1\end{bmatrix} = (1) \times 5 \times (4) + (-1) \times 2 \times (26) + (1) \times 1 \times 1 \times (3) = -29$$

Рис. 3.7. Вычисление детераминаната матрицы 3 x 3

Инверсии

Матрицы имеют аддитивные и мультипликативные инверсии.

Аддитивная инверсия

Аддитивная инверсия матрицы — это другая матрица B, такая, что A + B = 0. Другими словами, мы имеем элементы $b_{ij} = -a_{ij}$ для всех значений i и j. Обычно аддитивная инверсия A обозначается как (-A).

Мультипликативная инверсия

Мультипликативная инверсия определена только для квадратных матриц. Мультипликативная инверсия квадратной матрицы A — квадратная матрица B, такая, что $A \times B = B \times A = I$. Обычно мультипликативная инверсия обозначается как A^{-1}. Мультипликативная инверсия существует только, если det (A) имеет мультипликативную инверсию в соответствующем инверсном множестве. Если целое число не имеет мультипликативной инверсии в Z, то не существует мультипликативной инверсии матрицы в Z. Однако матрицы с реальными элементами имеют инверсии, только если $\det(A) \neq 0$.

Мультипликативные инверсии определены только для квадратных матриц.

Матрицы вычетов

Криптография использует матрицы вычетов: матрицы могут содержать все элементы из Z_n. Все операции на матрицах вычетов выполняются так же, как и на матрицах целых чисел, за исключением того, что операции производятся в модульной арифметике. Есть одно интересное свойство: матрица вычетов имеет мультипликативную инверсию, если детерминант матрицы имеет мультипликативную инверсию в Z_n.

Другими словами, матрица вычета имеет мультипликативную инверсию, если НОД (det (A), n) = 1.

Пример 3.7

Рисунок 3.8 показывает матрицу вычетов в Z_n и его мультипликативной инверсии A^{-1}. Возьмем детерминант det (A) = 21, который имеет мультипликативную инверсию 5 в Z_{26}. Обратите внимание, что когда мы умножаем эти две матрицы, то результат — единичная матрица мультипликативная матрица, в Z_{26}.

$$A = \begin{bmatrix} 3 & 5 & 7 & 2 \\ 1 & 4 & 7 & 2 \\ 6 & 3 & 9 & 17 \\ 13 & 5 & 4 & 16 \end{bmatrix} \qquad A^{-1} = \begin{bmatrix} 15 & 21 & 0 & 15 \\ 23 & 9 & 0 & 22 \\ 15 & 16 & 18 & 3 \\ 24 & 7 & 15 & 3 \end{bmatrix}$$

$$\det(A) = 21 \qquad \det(A^{-1}) = 5$$

Рис. 3.8. Матрица вычетов и мультипликативная инверсия

Сравнение

Две матрицы, сравнимые по модулю n, записываются как $A \equiv B (\mathrm{mod} n)$, если они имеют одинаковое число строк и столбцов и все соответствующие элементы — сравнимые по модулю n. Другими словами, $A \equiv B (\mathrm{mod} n)$, если $a_{ij} \equiv b_{ij} (\mathrm{mod} n)$ для всех i и j.

3.2. Линейное уравнение

Криптография часто включает в себя решение уравнения или множества уравнений одной или более переменных с коэффициентом в Z_n. Этот раздел показывает, как решать уравнения с одним неизвестным, когда степень переменной равна 1 (линейное уравнение).

Линейные уравнения с одним неизвестным, содержащие сравнения

Давайте посмотрим, как решаются уравнения с одним неизвестным, содержащие сравнения, то есть уравнения $ax = b \pmod{n}$. Уравнение этого типа может не иметь ни одного решения или иметь ограниченное число решений. Предположим, что НОД $(a, n) = d$. Если $d \nmid b$, решение не существует. Если $d \mid b$, то имеется d решений.

Если $d \mid b$, то для того, чтобы найти решения, мы используем следующую стратегию.

1. Сократить уравнение, разделив обе стороны уравнения (включая модуль) на d.
2. Умножить обе стороны сокращенного уравнения на мультипликативную инверсию, чтобы найти конкретное решение x_0.
3. Общие решения будут $x = x_0 + k\ (n/d)$ для $k = 0, 1..., (d - 1)$.

Пример 3.8

Решить уравнение $10x \equiv 2 (\mathrm{mod} 15)$.

Сначала мы найдем НОД$(10, 15) = 5$. Полученное число 5 не делится на 2, решение отсутствует.

Пример 3.9

Решить уравнение $14x \equiv 12 (\mathrm{mod} 18)$.

Решение

Заметим, что НОД $(14, 18) = 2$. Поскольку 2 делит 12, мы имеем точно два решения, но сначала сократим уравнение:

$$14x \equiv 12(mod\ 18) \rightarrow 7x \equiv 6(mod\ 9) \rightarrow x \equiv 6(7^{-1})(mod\ 9)$$
$$x_0 \equiv 6(7^{-1})(mod\ 9) \equiv (6 \times 4)(mod\ 9) = 6$$
$$x_1 = x_0 + 1 \times (18/2) = 15$$

Оба решения, `6` и `15`, удовлетворяют уравнению сравнения, потому что $(14 \times 6) \bmod 18 = 12$, а также $(14 \times 15) \bmod 18 = 12$.

Пример 3.10

Решить уравнение $3x + 4 \equiv 6 \,(\mathrm{mod} 13)$.

Решение

Сначала мы приводим уравнение к форме $ax \equiv b(\mathrm{mod} n)$. Мы прибавляем `(-4)` к обеим сторонам (`4` аддитивная инверсия). Получим $3x \equiv 2(\mathrm{mod} 13)$. Поскольку `НОД (3, 13) = 1`, уравнение имеет только одно решение, $x_0 = (2 \times 3^{-1}) \bmod 13 = 18 \mod 13 = 5$. Мы можем видеть, что ответ удовлетворяет первоначальному уравнению: $3 \times 5 + 4 = 6 \,(\mathrm{mod} 13)$.

Система линейных уравнений, содержащих сравнения

Мы можем решить систему линейных уравнений с одним и тем же модулем, если матрица, сформированная из коэффициентов системы уравнений, имеет обратную матрицу. Для решения уравнения составляются три матрицы. Первая — квадратная матрица — формируется из коэффициентов уравнения. Вторая — матрица-столбец — составляется из переменных. Третья — матрица-столбец в правой стороне оператора сравнения — состоит из значения b_n. Мы можем это уравнение представить как произведение матриц. Если обе стороны сравнения умножить на мультипликативную инверсию первой матрицы, в результате мы получим решение системы уравнений, как это показано на рис. 3.9.

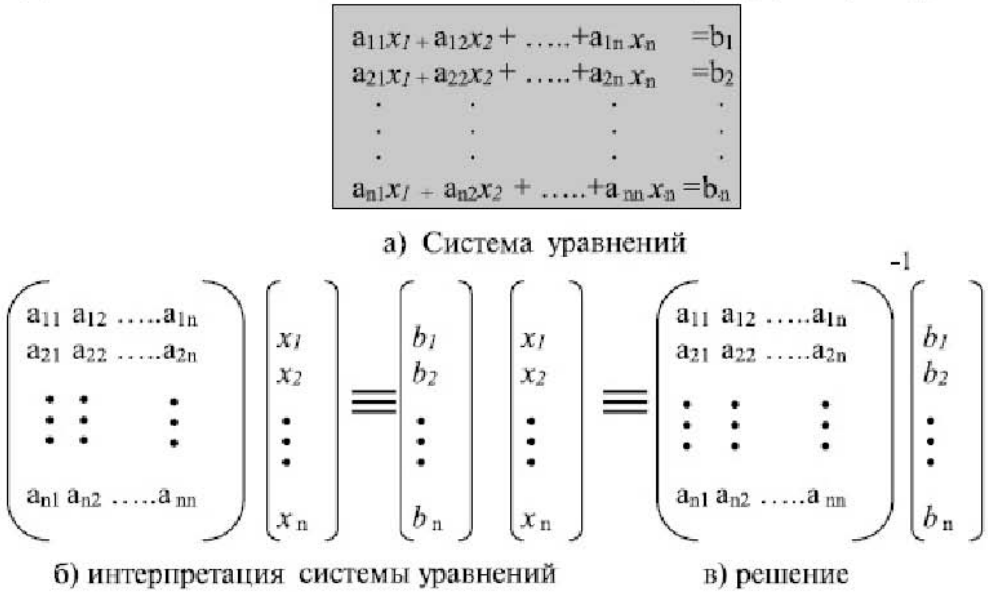


Рис. 3.9. Система линейных уравнений

Пример 3.11

Решить систему следующих трех уравнений:

$$3x + 5y + 7z = 3 \pmod{16}$$
$$x + 4y + 13z = 5 \pmod{16}$$
$$2x + 7y + 3z = 4 \pmod{16}$$

Решение

Здесь x, y и z играют роли x_1, x_2, и x_3. Матрица, сформированная из коэффициентов уравнений, — обратима. Мы находим мультипликативную инверсию матрицы и умножаем ее на матрицу столбца, сформированную из 3, 5 и 4. Результат — $x \equiv 15 (\mathrm{mod} 16)$, $y \equiv 4 (\mathrm{mod} 16)$ и $z \equiv 14 (\mathrm{mod} 16)$. Мы можем проверить ответ, подставляя эти значения в уравнения.

3.3. Рекомендованная литература

Для более детального изучения положений, обсужденных в этой

лекции, мы рекомендуем нижеследующие книги и сайты. Пункты, указанные в скобках, показаны в списке ссылок в конце книги.

Книги

Несколько книг дают простой, но полный охват теории чисел: [Ros06], [Sch99], [Cou99] и [BW00]. Матрицы обсуждаются в любой книге по линейной алгебре: [LEF04], [DF04] и [Dur05] — это хорошие книги для начинающих.

Сайты

Нижеследующие сайты дают больше информации о темах, рассмотренных в этой лекции.

- ссылка: http:en.wikipedia.org/wiki/Euclidean_algorithm - http:en.wikipedia.org/wiki/Euclidean_algorithm
- ссылка: http:en.wikipedia.org/wiki/Multiplicative_inverse - http:en.wikipedia.org/wiki/Multiplicative_inverse
- ссылка: http:en.wikipedia.org/wiki/Additive-inverse - http:en.wikipedia.org/wiki/Additive-inverse

3.4. Итоги

- Множество целых чисел, обозначаемое `Z`, содержит все целые числа от отрицательной бесконечности до положительной бесконечности. Для целых чисел определены три общих бинарных операции — сложение, вычитание и умножение. Деление не удовлетворяет определению бинарности, потому что требует два выхода вместо одного.
- В арифметике целых чисел, если мы делим `a` на `n`, мы можем получить `q` и `r`. Отношение между этими четырьмя целыми числами можно показать как $q \times n + r$. Мы говорим `a|n`, если $a = q \times n$. В этой лекции мы рассмотрели четыре свойства теории делимости.

- Два положительных целых числа могут иметь больше чем один общий делитель. Но мы обычно интересуемся наибольшим общим делителем. Алгоритм Евклида дает эффективный и систематический алгоритм вычисления наибольшего общего делителя двух целых чисел.
- Расширенный алгоритм Эвклида может вычислить НОД (a, b) и вычислить значение s и t, которые удовлетворяют уравнению as + bt = НОД (a, b). Линейное диофантово уравнение двух переменных: ax + by = c. Оно имеет частное и общие решения.
- В модульной арифметике мы интересуемся только остатками; мы хотим знать значение r, когда мы делим a на n. Мы используем новый оператор, названный модулем (mod), такой, что a mod n = r. Здесь n называется модулем, а r называется вычетом.
- Результат операции по модулю n — всегда целое число от 0 и до n-1. Мы можем сказать, что операция по модулю n создает набор, который в модульной арифметике называется множеством наименьших вычетов по модулю n, или Z_n.
- Отображение из Z в Z_n не совпадают один в один. Определенные элементы Z могут быть отображены в элемент Z_n. В модульной арифметике все целые числа в Z, отображаемые в Z_n, называются сравнениями по модулю. Для обозначения этой операции применяется оператор сравнения ($\equiv$).
- Система вычетов [a] — множество целых чисел, сравнимых по модулю n. Это множество всех целых чисел x = a (mod n).
- Три бинарных операции (сложение, вычитание и умножение), определенные для множества Z, могут быть также определены для множества Z_n. При необходимости результат может быть отражен в Z_n при помощи операции mod.
- В этой лекции для модульных операторов были определены несколько свойств.
- В Z_n два числа a и b — аддитивные инверсии по отношению друг к другу, если $a + b \equiv 0 (\mathrm{mod} n)$. Они — мультипликативные инверсии по отношению друг к другу, если $a \times b \equiv 1 (\mathrm{mod} n)$. Целое число a имеет мультипликативную инверсию в Z_n тогда и только тогда, когда НОД (n, a) = 1 (a

и n — взаимно простые числа).

- Расширенный алгоритм Евклида находит мультипликативные инверсии b в Z_n, когда даны n и b и НОД (n, b) = 1. Мультипликативная инверсия b — это значение t при соответствующем отображении в Z_n.
- Матрица — прямоугольный массив $l \times m$. элементы, где l является номером строки, а m — номер столбца. Мы обозначаем матрицу заглавной буквой и жирным шрифтом, например, A. Элемент a_{ij} расположен в i -той строке и j -том столбце.
- Две матрицы равны, если они имеют одинаковое число строк и столбцов и соответствующие элементы равны.
- Сложение и вычитание можно делать только для матриц равного размера. Мы можем умножить друг на друга две матрицы различных размеров, если число столбцов первой матрицы совпадает с числом строк второй матрицы. В матрицах вычетов все элементы берутся из Z_n.
- Все операции на матрицах вычетов проводятся в модульной арифметике.
- Матрица вычета имеет инверсию, если детерминант матрицы имеет инверсию.
- Уравнение $ax \equiv b (\mathrm{mod} n)$ не может иметь решения или ограниченное число решений. Если НОД (a,n)|b, то имеется ограниченное число решений.
- Система линейных уравнений с тем же самым модулем может быть решена, если матрица, сформированная из коэффициентов уравнений, имеет инверсию.

3.5. Набор для практики

Обзорные вопросы

1. Покажите различие между Z и Z_n. Какое из этих множеств может содержать отрицательные целые числа? Как мы можем отобразить целое число в Z в целое число в Z_n?
2. Перечислите четыре свойства теории делимости, обсужденной в

этой лекции. Приведите пример целого числа с единственным делителем. Приведите пример целого числа только с двумя делителями. Приведите пример целого числа с более чем двумя делителями.

3. Определите наибольший общий делитель двух целых чисел. Какой алгоритм может эффективно найти наибольший общий делитель?
4. Что такое линейное диофантово уравнение двух переменных? Сколько решений может иметь такое уравнение? Как может быть найдено решение(я)?
5. Что такое оператор по модулю и какие у него имеются приложения? Перечислите все свойства, которые мы упоминали в этой лекции для операций по модулю.
6. Определите сравнение и сопоставьте его свойства со свойствами равенства.
7. Определите систему вычетов и наименьший вычет.
8. Какова разница между множеством Z_n и множеством Z_{n*}? В каком множестве каждый элемент имеет аддитивную инверсию? В каком множестве каждый элемент имеет мультипликативную инверсию? Какой алгоритм используется, чтобы найти мультипликативную инверсию целого числа в Z_n?
9. Дайте определение матрицы. Что такое матрица-строка? Что такое матрица-столбец? Что такое квадратная матрица? Какая матрица имеет детерминант? Какая матрица может иметь инверсию?
10. Определите линейное сравнение. Какой алгоритм может использоваться, чтобы решить уравнение $ax \equiv b (\mathrm{mod} n)$? Как мы можем решить набор линейных уравнений?

Упражнения

1. Какие из следующих отношений являются истинными, а какие — ложными?

   5|26 3|123 27†127 15†21 23|96 8|5

2. Используя алгоритм Эвклида, найдите наибольший общий делитель следующих пар целых чисел:

- 88 и 220
- 300 и 42
- 24 и 320
- 401 и 700

3. Решите следующие примеры:
   - Дано НОД (a, b) = 24, найдите НОД (a, b, 16)
   - Дано НОД (a, b, c) = 12, найдите НОД (a, b, c, 16)
   - Найдите НОД (200, 180, и 450)
   - Найдите НОД (200, 180 450 610)
4. Предположим, что n — неотрицательное целое число.
   - Найдите НОД (2n + 1, n)
   - Используя результат части a, найдите НОД (201, 100), НОД (81, 40) и НОД (501, 250)
5. Предположим, что n — неотрицательное целое число.
   - Найдите НОД (3 n + 1,2n +1).
   - Используя результат части a, найдите НОД (301, 201) и НОД (121, 81)
6. Используя расширенный алгоритм Евклида, найдите наибольший общий делитель следующих пар и значения s и t:
   - 4 и 7
   - 291 и 42
   - 84 и 320
   - 400 и 60
7. Найдите результаты следующих операций:
   - 22 mod 7
   - 140 mod 10
   - -78 mod 13
   - 0 mod 15
8. Выполните следующие операции, сначала используя следующее сокращение:
   - (273 + 147) mod 10
   - (4223 + 17323) mod 10
   - (148 + 14432) mod 12
   - (2467+461) mod 12
9. Выполните следующие операции, сначала используя следующее сокращение:

- $(125 \times 45) \bmod 10$
- $(424 \times 32) \bmod 10$
- $(144 \times 34) \bmod 12$
- $(221 \times 23) \bmod 22$

10. Используя свойства оператора `mod`, докажите следующее:
    - Остаток от любого целого числа, когда оно делится на `10`, — самая правая цифра
    - Остаток от любого целого числа, когда оно делится на `100`, — целое число, составленное из двух самых правых цифр
    - Остаток от любого целого числа, когда оно делится на `1000`, — целое число, составленное из трех самых правых цифр
11. Из арифметики известно, что остаток от целого числа при делении на `5` — такой же, что и остаток от деления самой правой цифры на `5`. Используйте свойства оператора `mod`, чтобы доказать это утверждение.
12. Из арифметики известно, что остаток от целого числа при делении на `2` — такой же, что и остаток от деления самой правой цифры на `2`. Используйте свойства оператора `mod`, чтобы доказать это утверждение.
13. Из арифметики известно, что остаток от целого числа при делении на `4` — такой же, что и остаток от деления двух самых правых цифр на `4`. Используйте свойства оператора `mod`, чтобы доказать это утверждение.
14. Из арифметики известно, что остаток от целого числа при делении на `8` — такой же, что и остаток от деления самых правых трех цифр на `8`. Используйте свойства оператора `mod`, чтобы доказать это утверждение.
15. Из арифметики известно, что остаток от целого числа при делении на `9` — такой же, как и остаток от деления суммы его десятичных цифр на `9`. Другими словами, остаток от деления `6371` на `9` — такой же, как при делении `17` на `9`, потому что `6 + 3 + 7 + 1 = 17`. Используйте свойства оператора `mod`, чтобы доказать это утверждение.
16. Следующие упражнения показывают остатки от степени `10` при делении на `7`. Мы можем доказать, что эти значения будут

повторяться для более высоких степеней.

$$10^0 \bmod 7 = 1 \quad 10^1 \bmod 7 = 3 \quad 10^2 \bmod 7 = 2$$

$$10^3 \bmod 7 = 1 \quad 10^4 \bmod 7 = -3 \quad 10^5 \bmod 7 = -2$$

Используя вышеупомянутую информацию, найдите остаток от деления целого числа на 7. Проверьте ваш метод с числом 631453672.

17. Следующие упражнения показывают остатки от деления степеней 10 на 11. Мы можем доказать, что эти значения будут повторяться для более высоких степеней

$$10^2 \bmod 11 = 1 \quad 10^1 \bmod 11 = -1 \quad 10^2 \bmod 11 = 1 \quad 10 \bmod 11 = -1$$

Используя вышеупомянутую информацию, найдите остаток от деления целого числа на 11. Проверьте ваш метод с числом 631453672.

18. Следующие упражнения показывают остатки от деления степеней 10 на 13. Мы можем доказать, что эти значения будут повторяться для более высоких степеней.

$$10^2 \bmod 13 = 1 \quad 10^1 \bmod 13 = -3 \quad 10^2 \bmod 13 = -4$$

$$10^3 \bmod 3 = -1 \quad 10^4 \bmod 13 = 3 \quad 10^5 \bmod 13 = 4$$

Используя вышеупомянутую информацию, найдите остаток от целого числа при делении на 13. Проверьте ваш метод с числом 631453672.

19. Назначим числовые значения для заглавных букв латинского алфавита (A = 0, B = 1... Z = 25). Мы можем создать модульную арифметику, используя модуль 26.
    - Что является (A + N) mod 26 в этой системе?
    - Чему равно (A + 6) mod 26 в этой системе?
    - Чему равно (Y – 5) mod 26 в этой системе?

- Чему равно `(C - 10) mod 26` в этой системе?

20. Перечислите все пары аддитивной инверсии по модулю `20`.
21. Перечислите все мультипликативные обратные пары по модулю `20`.
22. Найдите мультипликативную инверсию каждого из следующих целых чисел в Z_{180}, используя расширенный алгоритм Евклида.
    - `38`
    - `7`
    - `132`
    - `24`
23. Найдите частное и общие решения следующих линейных диофантовых уравнений:
    - `25x + 10y = 15`
    - `19x + 13y = 20`
    - `14x + 21y = 77`
    - `40x +16y = 88`
24. Покажите, что нет ни одного решения следующих линейных диофантовых уравнений:
    - `15x + 12y = 13`
    - `18x + 30y = 20`
    - `15x + 25y = 69`
    - `40x +30y = 98`
25. Почтовое отделение продает марки только за `15` центов и за `39` центов. Найдите число марок, которые должен купить клиент, чтобы оплатить пересылку пакета стоимостью `2,70$`. Найдите несколько решений.
26. Найдите все решения каждого из следующих линейных уравнений:
    - $3x \equiv 4 \,(\mathrm{mod} 5)$
    - $4x \equiv 4 \,(\mathrm{mod} 6)$
    - $9x \equiv 12 \,(\mathrm{mod} 7)$
    - $256x \equiv 442 \,(\mathrm{mod} 60)$
27. Найдите все решения каждого из следующих линейных уравнений:
    - $3x + 5 \equiv 4 \,(\mathrm{mod} 5)$

- $4x + 6 \equiv 4 \,(\mathrm{mod} 6)$
- $9x + 4 \equiv 12 \,(\mathrm{mod} 7)$
- $232x + 42 \equiv 248 \,(\mathrm{mod} 50)$

28. Найдите $(A \times B) \bmod 16$, используя матрицы на рис. 3.11.

$$\underset{\mathbf{A}}{\begin{bmatrix} 3 & 7 & 10 \end{bmatrix}} \times \underset{\mathbf{B}}{\begin{bmatrix} 2 \\ 4 \\ 12 \end{bmatrix}} \qquad \underset{\mathbf{A}}{\begin{bmatrix} 3 & 4 & 6 \\ 1 & 1 & 8 \\ 5 & 8 & 3 \end{bmatrix}} \times \underset{\mathbf{B}}{\begin{bmatrix} 2 & 0 & 1 \\ 1 & 1 & 0 \\ 5 & 2 & 4 \end{bmatrix}}$$

Рис. 3.11. Матрицы для упражнения 28

29. На рисунке 3.12 найдите детерминант и мультипликативную инверсию для каждой матрицы вычетов в множестве Z_{10}.

$$\underset{\mathbf{A}}{\begin{bmatrix} 3 & 0 \\ 1 & 1 \end{bmatrix}} \qquad \underset{\mathbf{B}}{\begin{bmatrix} 4 & 2 \\ 1 & 1 \end{bmatrix}} \qquad \underset{\mathbf{C}}{\begin{bmatrix} 3 & 4 & 6 \\ 1 & 1 & 8 \\ 5 & 5 & 3 \end{bmatrix}}$$

Рис. 3.12. Матрицы для упражнения 29

30. Найдите все решения для следующих систем линейных уравнений:
    - $3x + 5y \equiv 4 \,(\mathrm{mod} 5)$ и $2x + y \equiv 3 \,(\mathrm{mod} 5)$
    - $3x + 2y \equiv 5 \,(\mathrm{mod} 7)$ и $4x + 6y \equiv 4 \,(\mathrm{mod} 7)$
    - $7x + 3y \equiv 3 \,(\mathrm{mod} 7)$ и $4x + 2y \equiv 5 \,(\mathrm{mod} 7)$
    - $2x + 5y \equiv 5 \,(\mathrm{mod} 8)$ и $x + 6y \equiv 3 \,(\mathrm{mod} 8)$

Традиционные шифры с симметричным ключом

Эта лекция представляет собой обзор традиционных шифров с симметричным ключом, которые использовались в прошлом. Изучение принципов таких шифров готовит читателя к следующим лекциям, которые рассматривают современные симметричные шифры. Эта лекция имеет несколько целей.

Общая идея шифров с симметричным ключом будет представлена с использованием примеров из криптографии. Вводимые термины и определения используются во всех более поздних лекциях, где речь пойдет о шифрах с симметричным ключом. Затем мы обсуждаем традиционные шифры с симметричным ключом. Эти шифры не используются сегодня, но мы изучаем их по нескольким причинам. Во-первых, они проще, чем современные шифры, и их легче понять. Во-вторых, они демонстрируют основы криптографии и шифрования. Эти основы могут использоваться для понимания современных шифров. В-третьих, рассказ о них подводит нас к изложению принципов построения и необходимости современных шифров, потому что традиционные шифры могут быть легко атакованы любыми пользователями компьютеров, и шифры, которые были безопасны в прежнее время, не обеспечивают безопасности при современном развитии компьютеров.

4.1. Введение

Рисунок 4.1 иллюстрирует общую идею шифра с симметричным ключом.

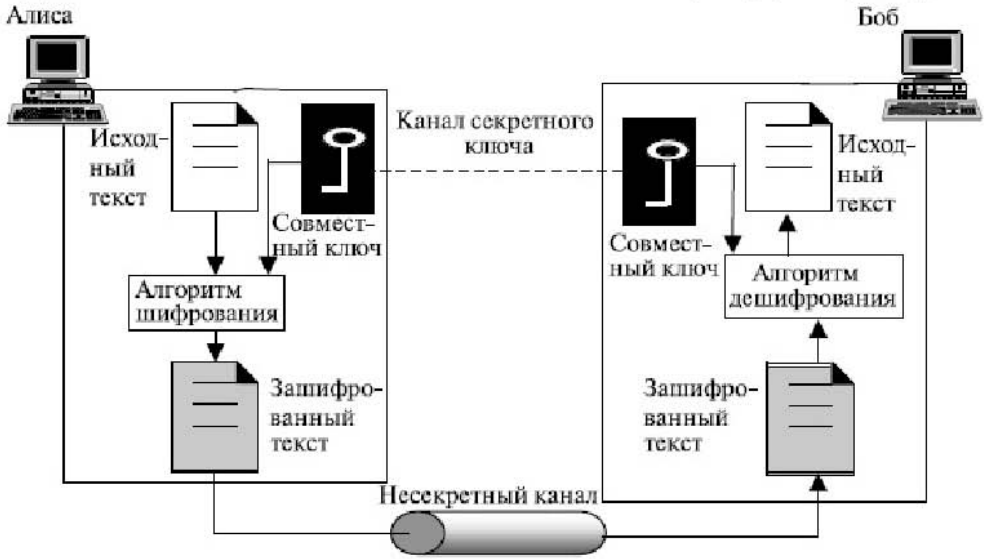


Рис. 4.1. Общая идея шифрования с симметричным ключом

На рисунке 4.1 объект, Алиса, может передать сообщение другому объекту, Бобу, по несекретному каналу, учитывая, что противник (назовем его Ева), не может понять содержание сообщения, просто подслушивая его по каналу.

Первоначальное сообщение от Алисы Бобу названо исходным текстом; сообщение, передаваемое через канал, названо зашифрованным текстом. Чтобы создать зашифрованный текст из исходного текста, Алиса использует алгоритм шифрования и совместный ключ засекречивания. Для того чтобы создать обычный текст из зашифрованного текста, Боб использует алгоритм дешифрования и тот же секретный ключ. Мы будем называть совместное действие алгоритмов шифрования и дешифрования шифровкой. Ключ — набор значений (чисел), которыми оперирует алгоритм шифровки.

Обратите внимание, что шифрование симметричными ключами использует единственный ключ (ключ, содержащий непосредственно набор кодируемых значений) и для кодирования и для дешифрования. Кроме того, алгоритмы шифрования и дешифрования — инверсии друг друга. Если P — обычный текст, C — зашифрованный текст, а K — ключ, алгоритм кодирования $E_K(x)$ создает зашифрованный текст из

исходного текста.

Алгоритм же дешифрования D_k (x) создает исходный текст из зашифрованного текста. Мы предполагаем, что E_k (x) и D_k (x) инверсны по отношению друг к другу. Они применяются, последовательно преобразуя информацию из одного вида в другой и обратно. Мы имеем

Шифрование: $C = E_k(P)$,
Расшифровка: $P = D_k(C)$,
где , $D_k(E_k(x)) = E_k(D_k(x)) = x$

Мы можем доказать, что исходный текст, созданный Бобом, тот же самый, что и исходный, переданный Алисой. Мы предполагаем, что Боб создает P_1 ; мы докажем, что $P_1 = P$:

Алиса: $C = E_k(P)$ Боб: $P_1 = D_k(C) = D_k(E_k(P)) = P$

Мы должны подчеркнуть, что согласно принципу Керкгоффса (приведенному далее) лучше делать алгоритм шифрования и дешифрования открытым, но сохранять в тайне совместный ключ. Это означает, что Алиса и Боб нуждаются в другом защищенном канале для обмена ключом засекречивания. Алиса и Боб могут однажды встретиться и обменяться ключом лично. Защищенный канал аналогично представляет собой "встречу лицом к лицу" для обмена ключом. Они могут также довериться третьему лицу, чтобы он дал им одинаковые ключи. Они могут создать временный ключ засекречивания, используя другой вид асимметрично-ключевых шифров, который будет рассмотрен в более поздних лекциях. В этой лекции мы просто принимаем, что существует утвержденный ключ засекречивания между Алисой и Бобом.

Применяя шифрование симметричными ключами, Алиса и Боб могут использовать тот же самый ключ для связи на другом направлении, от Боба к Алисе. Именно поэтому метод назван симметричным.

Другой элемент в шифровании симметричными ключами — число ключей. Алиса нуждается в другом ключе засекречивания, чтобы связаться с другим человеком, скажем, Дэвидом. Если есть m группа

людей, в которой каждый должен иметь связь друг с другом, сколько ключей необходимо? Ответ — $(m \times (m-1))/2$, потому что каждому человеку надо `m - 1` ключ, чтобы связаться с остальной частью группы, но ключ между `A` и `B` может использоваться в обоих направлениях. В более поздних лекциях мы увидим, как решается эта проблема.

Шифрование можно представлять себе как замок, который запирает ящик, содержащий сообщение; дешифрование можно представлять себе как открытие замка такого ящика. В шифровании симметричными ключами один и тот же ключ замыкает и размыкает замок, как это показано на рис. 4.2. В более поздних лекциях будет рассказано, что шифрование асимметричными ключами нуждается в двух ключах: одном для замыкания, а втором — для размыкания замка.

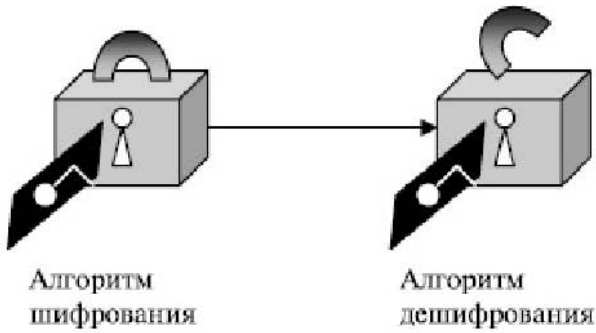


Рис. 4.2. Шифрования симметричными ключами, как замыкание и размыкание замка с тем же самым ключом

Принципы Керкгоффса

Хотя можно предположить, что шифр был бы более безопасен, если мы скрываем и алгоритм шифрования/дешифрования, и ключ засекречивания, это не рекомендуется. Согласно принципу Керкгоффса, нужно всегда предполагать, что противник — Ева — знает алгоритм кодирования/дешифрования. Противодействие шифра атаке должно базироваться только на тайне ключа. Другими словами, предполагается, что ключ должен быть настолько труден, что не надо скрывать алгоритм

кодирования/дешифрования. Эти принципиальные положения станут более ясны, когда мы будем изучать современные шифры. Для современных шифров сегодня существует немного алгоритмов. Множество ключей (Ключевой домен) для каждого алгоритма, однако, настолько большое число, что мешает противнику найти ключ.

Криптоанализ

Криптография — наука и искусство создания секретных кодов, криптоанализ — наука и искусство взламывания этих кодов. В дополнение к изучению методов криптографии мы также должны изучить методы криптоанализа.

Это необходимо не для того, чтобы взламывать коды других людей, а чтобы оценить уязвимые места наших криптографических систем. Изучение криптоанализа помогает нам создавать лучшие секретные коды. Есть четыре общих типа атак криптоанализа, показанные на рис. 4.3. В этой и следующих лекциях мы будем разбирать некоторые из этих атак на конкретные шифры.

Рис. 4.3. Атаки криптоанализа

Атака только на зашифрованный текст

В атаке только на зашифрованный текст Ева имеет доступ только к некоторому зашифрованному тексту. Она пробует найти соответствующий ключ и исходный текст. При этом, согласно предположению, Ева знает алгоритм и может перехватить зашифрованный текст. Атака только зашифрованного текста — самая

вероятная, потому что Еве для нее нужен только сам текст. Шифр должен серьезно препятствовать этому типу атаки и не позволить дешифрование сообщения противником. Рисунок 4.4 иллюстрирует процесс атаки.

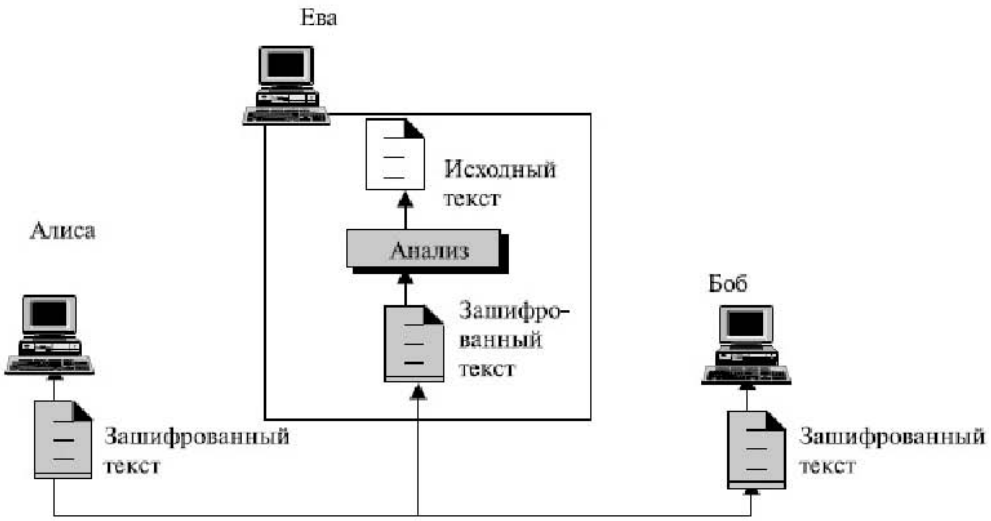


Рис. 4.4. Атака только на зашифрованный текст

В атаке только на зашифрованный текст могут использоваться различные методы. Мы рассмотрим здесь только некоторые из них.

Атака грубой силы

При методе грубой силы, или методе исчерпывающего ключевого поиска, Ева пробует использовать все возможные ключи. Мы предполагаем, что Ева знает алгоритм и знает множество ключей (список возможных ключей). При использовании этого метода перехватывается исходный текст и задействуются все возможные ключи, пока не получится исходный текст. Создание атаки грубой силы было в прошлом трудной задачей; сегодня с помощью компьютера это стало проще. Чтобы предотвратить этот тип атаки, число возможных ключей должно быть очень большим.

Статистическая атака

Криптоаналитик может извлечь выгоду из некоторых свойственных языку исходного текста характеристик, чтобы начать статистическую атаку. Например, мы знаем, что буква E — наиболее часто используемая буква в английском тексте. Криптоаналитик находит наиболее часто используемый символ в зашифрованном тексте и принимает, что это соответствующий символ исходного текста — E. После определения нескольких пар аналитик может найти ключ и расшифровать сообщение. Чтобы предотвратить этот тип атаки, шифр должен скрывать характеристики языка.

Атака по образцу

Некоторые шифры скрывают характеристики языка, но создают некоторые образцы в зашифрованном тексте. Криптоаналитик может использовать атаку по образцу, чтобы взломать шифр. Поэтому важно использовать шифры, которые сделали бы просматриваемый зашифрованный текст насколько возможно неопределенным, абстрактным.

Атака знания исходного текста

При атаке знания исходного текста Ева имеет доступ к некоторым парам "исходный/зашифрованный текст" в дополнение к перехваченному зашифрованному тексту, который она хочет взломать, как показано на рис. 4.5.

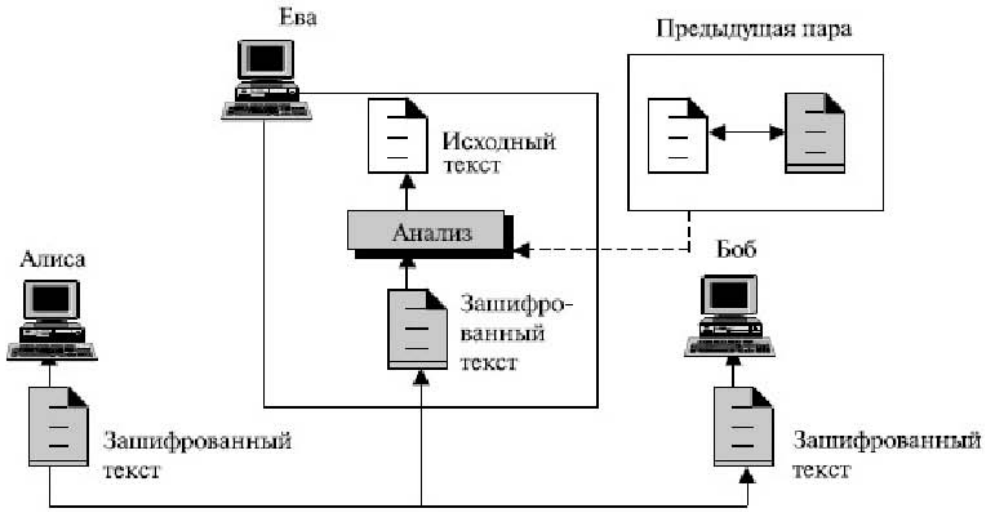


Рис. 4.5. Атака знания исходного текста

Пары исходного/зашифрованного текста были собраны ранее. Например, Алиса передала секретное сообщение Бобу, но она позже открыла содержание сообщения посторонним. Ева хранила и зашифрованный текст, и исходный текст, чтобы использовать их, когда понадобится взломать следующее секретное сообщение от Алисы Бобу, предполагая, что Алиса не изменит свой ключ. Ева использует отношения между предыдущей парой, чтобы анализировать текущий зашифрованный текст. Те же самые методы, используемые в атаке только для зашифрованного текста, могут быть применены здесь. Но эту атаку осуществить проще, потому что Ева имеет больше информации для анализа. Однако может случиться, что Алиса изменила свой ключ или не раскрывала содержания любых предыдущих сообщений, — тогда подобная атака станет невозможной.

Атака с выборкой исходного текста

Атака с выборкой исходного текста подобна атаке знания исходного текста, но пары "исходный/зашифрованный текст" были выбраны и изготовлены самим нападавшим. Рисунок 4.6 иллюстрирует этот процесс.

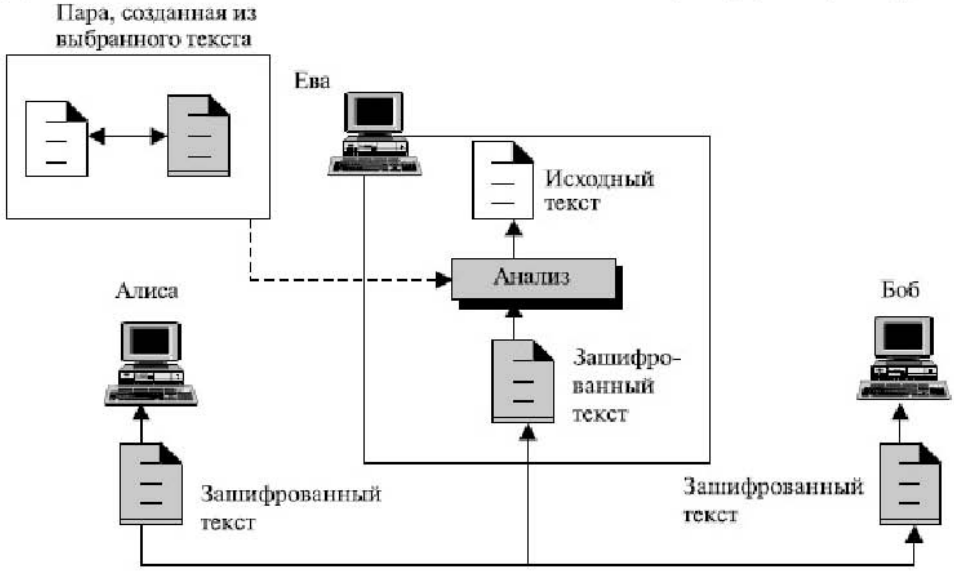


Рис. 4.6. Атака с выборкой исходного текста

Это может случиться, например, если Ева имеет доступ к компьютеру Алисы. Она выбирает некоторый исходный текст и создает с помощью компьютера зашифрованный текст. Конечно, она не имеет ключа, потому что ключ обычно размещается в программном обеспечении, используемом передатчиком. Этот тип атаки намного проще осуществить, но он наименее вероятен, поскольку подразумевает слишком много "если".

Атака с выбором зашифрованного текста

Атака с выбором зашифрованного текста подобна атаке с выборкой исходного текста, за исключением того, что выбирает некоторый зашифрованный текст и расшифровывает его, чтобы сформировать пару "зашифрованный/исходный текст" (это случается, когда Ева имеет доступ к компьютеру Боба). Рисунок 4.7 показывает этот процесс.

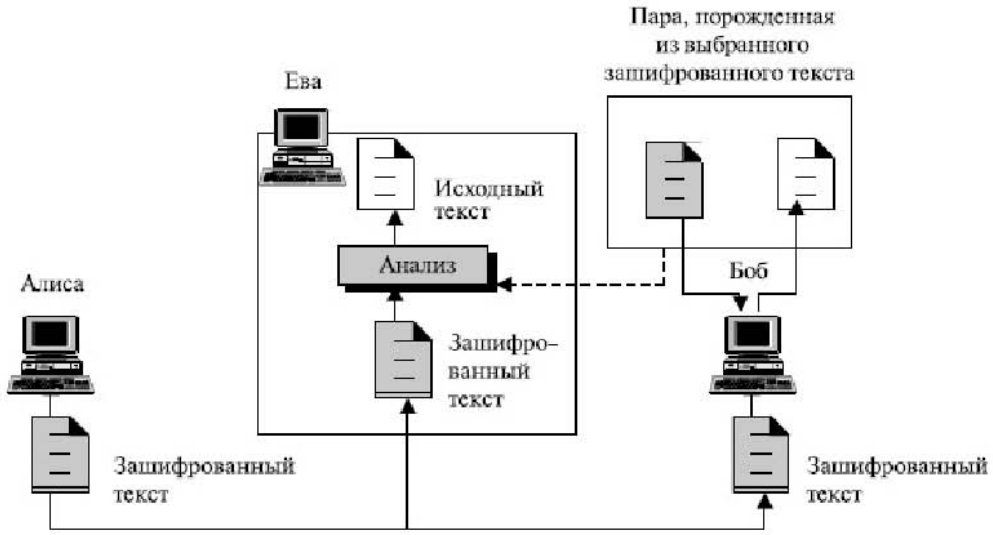


Рис. 4.7. Атака с выбором зашифрованного текста

Категории традиционных шифров

Мы можем разделить традиционные шифры с симметричным ключом на две обширные категории: шифры подстановки и шифры перестановки. В шифре подстановки мы заменяем один символ в зашифрованном тексте на другой символ; в шифре перестановки — меняем местами позиции символов в исходном тексте.

4.2. Шифры подстановки

Шифр подстановки заменяет один символ другим. Если символы в исходном тексте — символы алфавита, мы заменяем одну букву другой. Например, мы можем заменить букву `A` буквой `D`, а букву `T` — буквой `Z`. Если символы — цифры (от `0` до `9`), мы можем заменить `3` на `7` и `2` на `6`. Шифры подстановки могут быть разбиты на две категории: моноалфавитные или многоалфавитные шифры.

Шифр подстановки заменяет один символ другим.

Моноалфавитные шифры

Сначала обсудим шифры подстановки, называемые моноалфавитными шифрами. В такой подстановке буква (или символ) в исходном тексте всегда изменяется на одну и ту же самую букву (или символ) в зашифрованном тексте независимо от его позиции в тексте. Например, если алгоритм определяет, что буква A в исходном тексте меняется на букву D, то при этом каждая буква A изменяется на букву D. Другими словами, буквы в исходном тексте и зашифрованном тексте находятся в отношении один к одному.

В моноалфавитной подстановке отношения между буквой в исходном тексте и буквой в зашифрованном тексте — один к одному.

Пример 4.1

Приведенный ниже пример показывает исходный текст и соответствующий ему зашифрованный текст. Мы используем строчные символы, чтобы показать исходный текст, и заглавные буквы (символы верхнего регистра), чтобы получить зашифрованный текст. Шифр моноалфавитный, потому что оба l зашифрованы как O.

Исходный текст: hello Зашифрованный текст: KHOOR

Пример 4.2

Приведенный ниже пример показывает исходный текст и соответствующий ему зашифрованный текст. Шифр не является моноалфавитным, потому что каждая буква l (эль) зашифрована различными символами. Первая буква l (эль) зашифрована как N ; вторая — как Z.

Исходный текст: hello Зашифрованный текст: ABNZF

Аддитивный шифр

Самый простой моноалфавитный шифр — аддитивный шифр, его иногда называют шифром сдвига, а иногда — шифром Цезаря, но

термин аддитивный шифр лучше показывает его математический смысл. Предположим, что исходный текст состоит из маленьких букв (от a до z) и зашифрованный текст состоит из заглавных букв (от A до Z). Чтобы обеспечить применение математических операций к исходному и зашифрованному текстам, мы присвоим каждой букве числовое значение (для нижнего и верхнего регистра), как это показано на рис. 4.8.

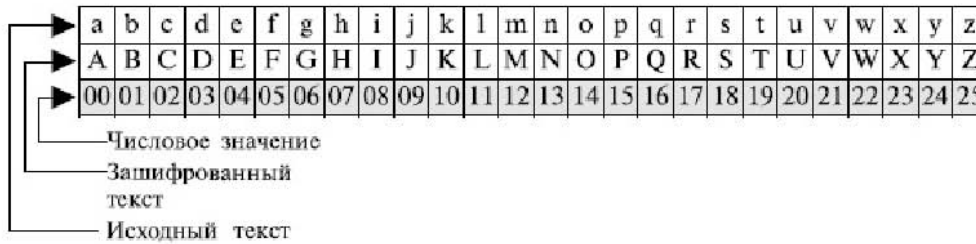


Рис. 4.8. Представление букв исходного текста и зашифрованного текста в Z26

На рисунке 4.8 каждому символу (нижний регистр или верхний регистр) сопоставлено целое число из Z_{26}. Ключ засекречивания между Алисой и Бобом — также целое число в Z_n. Алгоритм кодирования прибавляет ключ к символу исходного текста; алгоритм дешифрования вычитает ключ из символа зашифрованного текста. Все операции проводятся в Z_n. Рисунок 4.9 показывает процесс шифрования и дешифрования.

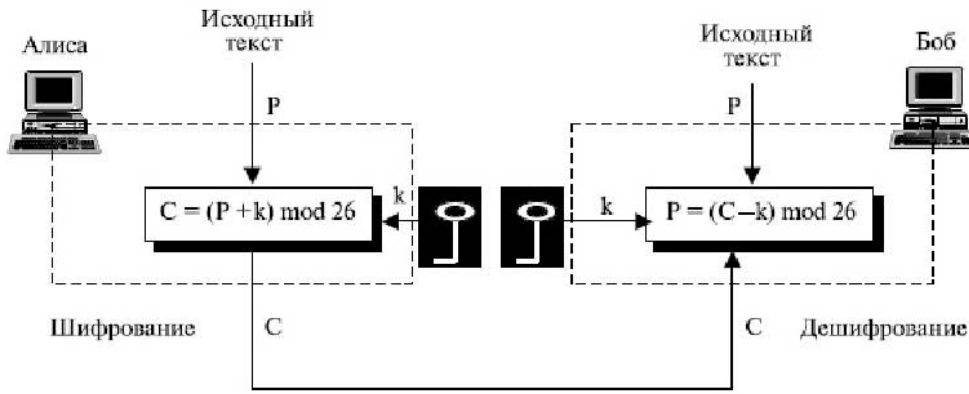


Рис. 4.9. Аддитивный шифр

Мы можем легко показать, что шифрование и дешифрование являются инверсными друг другу, потому что исходный текст, созданный Бобом (P_1), тот же самый, что и тот, который передан Алисой (P).

$$P_1 = (C - k) \bmod 26 = (P + k - k) \bmod 26 = P$$

Когда применяется аддитивный шифр, исходный текст, зашифрованный текст и ключ — целые числа в Z_{26} .

Пример 4.3

Используйте аддитивный шифр с ключом = 15, чтобы зашифровать сообщение "hello".

Решение

Мы применяем алгоритм кодирования к исходному тексту, буква за буквой:

```
Исходный текст h ->   07  Шифрование (07 + 15) mod 26    Шифр. Тек
Исходный текст e ->   04  Шифрование (04+ 15)  mod 26    Шифр. Тек
Исходный текст l ->   11  Шифрование (11 + 15) mod 26    Шифр. Тек
Исходный текст l ->   11  Шифрование (11 + 15) mod 26    Шифр. Тек
Исходный текст o ->   14  Шифрование (14 + 15) mod 26    Шифр. Тек
```

Результат — "WTAAD". Обратите внимание, что шифр моноалфавитный, потому что два отображения одной и той же буквы исходного текста (l) зашифрованы как один и тот же символ (A).

Пример 4.4

Используйте шифр сложения с ключом = 15, чтобы расшифровать сообщение "WTAAD".

Решение

Мы применяем алгоритм дешифрования к исходному тексту буква за буквой:

```
Шифр. Текст   W ->  22    Шифрование (22 - 15) mod 26    Исходный
```

```
Шифр. Текст   T ->  19    Шифрование (19 - 15) mod 26    Исходный
Шифр. Текст   A ->  00    Шифрование (00 - 15) mod 26    Исходный
Шифр. Текст   A ->  00    Шифрование (00 - 15) mod 26    Исходный
Шифр. Текст   D ->  03    Шифрование (03 - 15) mod 26    Исходный
```

Результат — "hello". Обратите внимание, что операции проводятся по модулю 26 (см. лекции 2-3), отрицательный результат должен быть отображен в Z_{26} (например, −15 становится 11).

Шифр сдвига

Исторически аддитивные шифры назывались шифрами сдвига — по той причине, что алгоритм шифрования может интерпретироваться как "клавиша сдвига буквы вниз", а алгоритм дешифрования может интерпретироваться как "клавиши сдвига буквы вверх". Например, если ключ = 15, алгоритм кодирования сдвигает букву на 15 букв вниз (к концу алфавита). Алгоритм дешифрования сдвигает букву на 15 букв вверх (к началу алфавита). Конечно, когда мы достигаем конца или начала алфавита, мы двигаемся по кольцу к началу (объявленные свойства операции по модулю 26).

Шифр Цезаря

Юлий Цезарь использовал аддитивный шифр, чтобы связаться со своими чиновниками. По этой причине аддитивные шифры упоминаются иногда как шифры Цезаря. Цезарь для своей связи использовал цифру 3.

Аддитивные шифры упоминаются иногда как шифры сдвига или шифры Цезаря.

Криптоанализ

Аддитивные шифры уязвимы к атакам только зашифрованного текста, когда используется исчерпывающий перебор ключей (атака грубой силы). Множество ключей аддитивного шифра очень мало — их только

26. Один из ключей, нулевой, является бесполезным (зашифрованный текст будет просто соответствовать исходному тексту). Следовательно, остается только 25 возможных ключей. Ева может легко начать атаку грубой силы зашифрованного текста.

Пример 4.5

Ева перехватила зашифрованный текст "UVACLYFZLJBYL". Покажите, как она может взломать шифр, используя атаку грубой силы.

Решение

Ева пробует раскрыть текст и последовательно перебирает ключи начиная с первого. С помощью ключа номер 7 она получает осмысленный текст "not very secure" (не очень безопасный).

Зашифрованный текст: UVACLYFZLJBYL
K = 1 Исходный текст: tubkxeykiaxk
K = 2 Исходный текст: styajwdxjhzwj
K = 3 Исходный текст: rsxzivewigyvi
K = 4 Исходный текст: qrwyhubvhfhuh
K = 5 Исходный текст: pqvxgtaugewtg
K = 6 Исходный текст: opuwfsztfdvst
K = 7 Исходный текст: notverysecure

Статистические атаки

Аддитивные шифры также могут быть объектами статистических атак. Это особенно реально, если противник перехватил длинный зашифрованный текст. Противник может воспользоваться знаниями о частоте употребления символов в конкретном языке. Таблица 4.1 показывает частоту появления определенных букв для английского текста длиной в 100 символов.

Таблица 4.1. Частота появления букв в английском тексте

Буква	Частота	Буква	Частота	Буква	Частота	Буква	Частота
E	12,7	H	6,1	W	2,3	K	0,08
T	9,1	R	6,0	F	2,2	J	0,02

A	8,2	D	4,3	G	2,0	Q	0,01
O	7,5	L	4,0	Y	1,9	X	0,01
I	7,0	C	2,8	P	1,5	Z	0,01
N	6,7	U	2,8	B	1,0		
S	6,3	M	2,4	V			

Однако информации о частоте единственного символа недостаточно, и это затрудняет анализ шифрованного текста, основанный на анализе частоты появления букв. Весьма желательно знать частоту появления комбинаций символов. Мы должны знать частоту появления в зашифрованном тексте комбинаций с двумя или с тремя символами и сравнивать ее с частотой в языке, на котором написан исходный документ.

Наиболее употребляемые группы с двумя символами (диаграмма (diagrams)) и группы с тремя символами (триграмма (trigrams)) для английского текста показаны в таблице 4.2.

Таблица 4.2. Группы диаграмм и триграмм , основанные на их час

Диаграмма	TH,HE,IN,ER,AN,RE,ED,ON,ES,ST,EN,AT,TO,NT,HA,ND,OU,
Триграмма	THE,ING,AND,HER,ERE,ENT,THA,NTH,WAS,ETH,FOR,DTH

Пример 4.6

Ева перехватила следующий зашифрованный текст. Используя статистическую атаку, найдите исходный текст.

XLILSYWIMWRSAJSVWEPIJSVJSYVQMPPMSRHSPPEVWMXMWAS

Решение

Когда Ева составит таблицу частоты букв в этом зашифрованном тексте, она получит: `I = 14, V = 13, S = 12`, и так далее. Самый частый символ – `I` — имеет `14` появлений. Это показывает, что символ `I` в зашифрованном тексте, вероятно, соответствует символу `e` в исходном тексте. Тем самым, ключ `= 4`. Ева расшифровывает текст и получает

the house is now for sale for four million dollars it is worth more hurry before the
дом теперь продается за четыре миллиона долларов, стоит поспешить,

Мультипликативные шифры

В мультипликативном шифре алгоритм шифрования применяет умножение исходного текста ключом, а алгоритм дешифрования применяет деление зашифрованного текста ключом, как показано на рис. 4.10. Однако поскольку операции проводятся в Z_{26}, дешифрование здесь означает умножение на мультипликативную инверсию ключа. Обратите внимание, что ключ должен принадлежать набору Z_{n*} — это гарантирует, что шифрование и дешифрование инверсны друг другу.

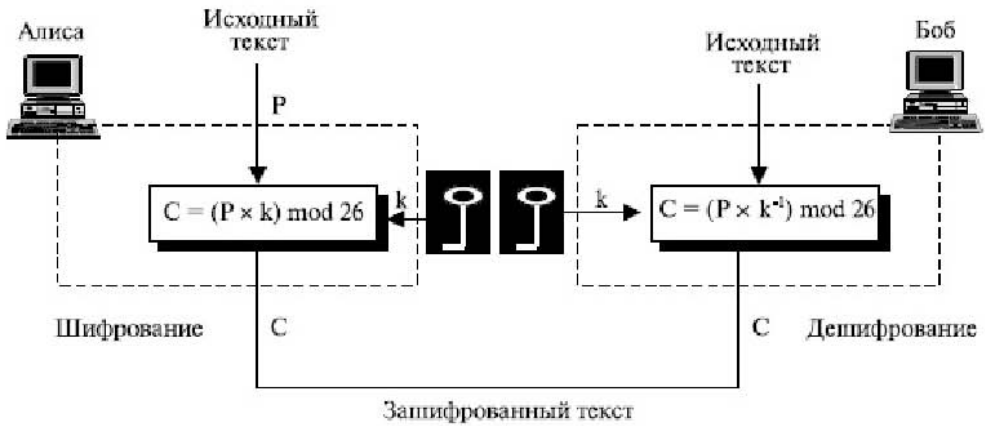


Рис. 4.10. Мультипликативный шифр

В мультипликативном шифре исходный текст и зашифрованный текст — целые числа в Z_n; ключ — целое число в Z_{n*}.

Пример 4.7

Каково множество ключей для любого мультипликативного шифра?

Решение

Ключ должен быть в Z_{26*}. Это множество имеет только 12 элементов: 1, 3, 5, 7, 9, 11, 15, 17, 19, 21, 23, 25.

Пример 4.8

Мы используем мультипликативный шифр, чтобы зашифровать сообщение "hello" с ключом 7. Зашифрованный текст "XCZZU".

Исходный текст	h	07	Шифрование (07 x 07) mod 26	Шифр. Текс
Исходный текст	e	04	Шифрование (04 x 07) mod 26	Шифр. Текс
Исходный текст	l	11	Шифрование (11 x 07) mod 26	Шифр. Текс
Исходный текст	l	11	Шифрование (11 x 07) mod 26	Шифр. Текс
Исходный текст	o	14	Шифрование (14 x 07) mod 26	Шифр. Текс

Мы можем комбинировать аддитивные и мультипликативные шифры, чтобы получить то, что названо аффинным шифром — комбинацией обоих шифров с парой ключей. Первый ключ используется мультипликативным шифром, второй — аддитивным шифром. Рисунок 4.11 доказывает, что афинный шифр — фактически два шифра, применяемые один за другим. Мы могли бы показать только одну комплексную операцию для шифрования или дешифрования, такую, как

$$C = (P \times k_1 + k_2) \bmod 26 \quad \text{и} \quad P = ((C - k_2) \times {k_1}^{-1}) \bmod 26.$$

Однако мы использовали временный результат (T) и указали две отдельных операции, показав тем самым, что всякий раз, когда мы используем комбинацию шифров, нужно убедиться, что каждый из них имеет инверсию на другой стороне линии и что они используются в обратном порядке в шифровании и дешифровании. Если сложение — последняя работа в шифровании, то вычитание должно быть первым в дешифровании.

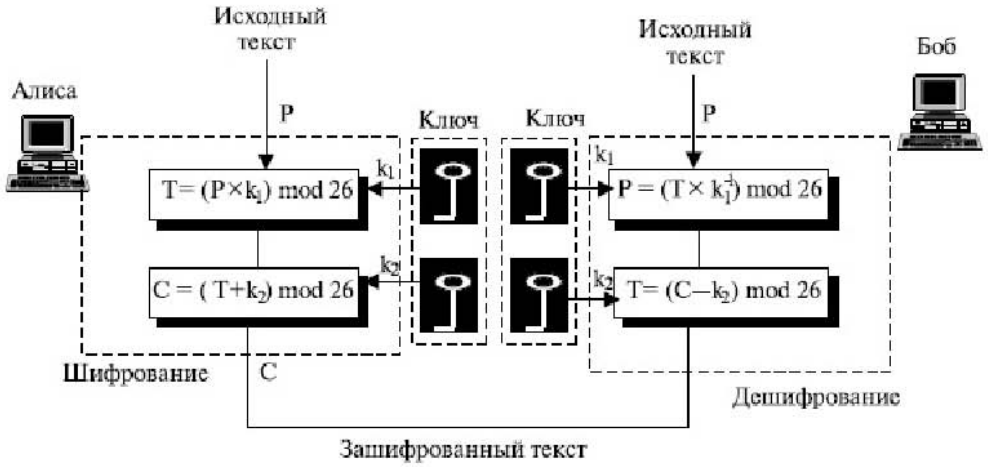


Рис. 4.11. Аффинный шифр

При аффинном шифре отношение между исходным текстом P и шифрованным текстом C определяется, как это показано ниже.

$C = (P \times k_1 + k_2) \bmod 26 \quad P = ((C - k_2) \times k_1^{-1}) \bmod 26$

где k_1^{-1} мультипликативная инверсия k_1, а$(-k_2)$–аддитивная инверсия k

Пример 4.9

Аффинный шифр использует пару ключей, в которой первый ключ из Z_{26}^*, а второй — из Z_{26}. Область существования ключей равна $26 \times 12 = 312$.

Пример 4.10

Используйте аффинный шифр, чтобы зашифровать сообщение "hello" с ключевой парой (7, 2).

Решение

Мы используем 7 для мультипликативного ключа и 2 для аддитивного ключа. Получаем "ZEBBW".

P: h	07	Шифрование (07 x 07+2) mod 26	C: 25	Z
P: e	04	Шифрование (04 x 07+2) mod 26	C: 04	E

P: l	11	Шифрование (11 x 07+2) mod 26	C: 01	B
P: l	11	Шифрование (11 x 07+2) mod 26	C: 01	B
P: o	14	Шифрование (14 x 07+2) mod 26	C: 22	W

Пример 4.11

Используйте аффинный шифр, чтобы расшифровать сообщение "ZEBBW" с ключевой парой (7, 2) в модуле 26.

Решение

Чтобы найти символы исходного текста, прибавим аддитивную инверсию от $(-2) \equiv 24 \pmod{26}$ к полученному зашифрованному тексту. Потом умножим результат на мультипликативную инверсию от $7^{-1} \equiv 15 \pmod{26}$. Поскольку 2 имеет аддитивную инверсию в Z_{26} и 7 имеет мультипликативную инверсию в Z_{26^*}, исходный текст — точно тот, что мы использовали в примере 4.10.

C: 25	->	Z	Дешифрование (07 x 07 - 2) mod 26	P: 07	-> h
C: 04	->	E	Дешифрование (04 x 07 - 2) mod 26	P: 04	-> e
C: 01	->	B	Дешифрование (11 x 07 - 2) mod 26	P: 11	-> l
C: 01	->	B	Дешифрование (11 x 07 - 2) mod 26	P: 11	-> l
C: 22	->	W	Дешифрование (14 x 07 - 2) mod 26	P: 14	-> o

Пример 4.12

Аддитивный шифр — частный случай аффинного шифра, при котором $k_1 = 1$. Мультипликативный шифр — частный случай аффинного шифра, в котором $k_2 = 0$.

Криптоанализ аффинного шифра

Хотя методы грубой силы и статистический метод атаки могут использоваться только для зашифрованного текста, попробуем атаку с выборкой исходного текста. Предположим, что Ева перехватывает следующий зашифрованный текст:

PWUFFOGWCHFDWIWEJOUUNJORSMDWRHVCMWJUPVCCG

Ева также очень ненадолго получает доступ к компьютеру Алисы и время, достаточное лишь для того, чтобы напечатать исходный текст с двумя символами: "et". Тогда она пробует зашифровать короткий исходный текст, используя два различных алгоритма, потому что не уверена, какой из них является аффинным шифром.

Для того чтобы найти ключ, Ева использует следующую стратегию:

a. Ева знает, что если первый алгоритм является аффинным, она может составить следующие уравнения, основанные на первом наборе данных:

e -> W 04 -> 22 $(04 \times k_1 + k_2) = 22 \pmod{26}$
t- -> C 19 -> 02 $(19 \times k_1 + k_2) = 02 \pmod{26}$

Как мы узнали в лекции 4, эти два уравнения сравнения могут быть решены (могут быть найдены значения k_1, и k_2). Однако этот ответ неприемлем, потому что $k_1 = 16$ не может быть первой частью ключа. Его значение, 16, не имеет мультипликативной инверсии в Z_{26^*}.

$$\begin{pmatrix} k_1 \\ k_2 \end{pmatrix} = \begin{pmatrix} 4 & 1 \\ 19 & 1 \end{pmatrix}^{-1} \cdot \begin{pmatrix} 22 \\ 2 \end{pmatrix} = \begin{pmatrix} 19 & 7 \\ 3 & 24 \end{pmatrix} \cdot \begin{pmatrix} 22 \\ 2 \end{pmatrix} = \begin{pmatrix} 161 \\ 10 \end{pmatrix} \rightarrow k_1 = 16 \; k_2 = 10$$

b. Ева теперь пробует использовать результат второго набора данных:

e -> W 04 -> 22 $(04 \times k_1 + k_2) = 22 \pmod{26}$
t- -> C 19 -> 05 $(19 \times k_1 + k_2) = 05 \pmod{26}$

Квадратная матрица и ее инверсия — те же самые, что и в предыдущем примере. Теперь Ева получает $k_1 = 11$ и $k_2 = 4$, эта пара является приемлемой, потому что k_1 имеет мультипликативную инверсию в Z_{26^*}. Она пробует пару ключей (19, 22), которые являются инверсией пары (11, 4), и расшифровывает сообщение. Исходный текст

Best time of the year is spring when flower bloom

Самое лучшее время года — весна, когда цветут цветы

Моноалфавитный шифр подстановки

Поскольку аддитивные, мультипликативные и аффинные шифры имеют малое множество ключей, они очень уязвимы к атаке грубой силы. Алиса и Боб согласовали единственный ключ, который они используют, чтобы зашифровать каждую букву в исходном тексте или расшифровать каждую букву в зашифрованном тексте. Другими словами, ключ независим от передаваемых букв.

Лучшее решение состоит в том, чтобы создать отображение каждой буквы исходного текста на соответствующий символ зашифрованного текста. Алиса и Боб могут договориться об отображении для каждой буквы и записать его в виде таблицы. Рисунок 4.12 показывает пример такого отображения.

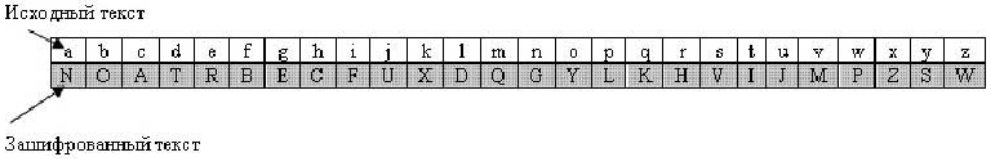


Рис. 4.12. Пример ключа для моноалфавитного шифра подстановки

Пример 4.13

Мы можем использовать ключ, показанный на рисунке 4.12, чтобы зашифровать сообщение

This message is easy to encrypt but hard to find the key
(это сообщение просто зашифровать, но трудно найти ключ, которым за

Зашифрованное сообщение имеет вид

ICFVQRVVNEFVRNVSIYRGAHSLIOJICNHTIYBFGTICRXRS

Криптоанализ

Размер ключевого пространства для моноалфавитного шифра подстановки — число перестановок из 26, т.е. 26! (почти 4×10^{26}). Это делает атаку грубой силы чрезвычайно трудной для Евы, даже если она использует мощный компьютер. Однако она может применить статистическую атаку, основанную на частоте символов. Шифр не изменяет частоту употребления символов.

Моноалфавитные шифры не изменяют частоту появления символов в зашифрованном тексте, что делает шифры уязвимыми к статистической атаке.

Многоалфавитные шифры

В многоалфавитной подстановке каждое появление символа может иметь различную замену. Отношения между символом в исходном тексте и символом в зашифрованном тексте — "один ко многим". Например, "a" может быть зашифровано как "D" в начале текста, но как "N" — в середине. Многоалфавитные шифры имеют преимущество: они скрывают частоту появления символа основного языка. Ева не может использовать статистическую частоту отдельного символа, чтобы взломать зашифрованный текст.

Чтобы создать многоалфавитный шифр, мы должны сделать каждый символ зашифрованного текста зависящим от соответствующего символа исходного текста и позиции символа исходного текста в сообщении. Это подразумевает, что наш ключ должен быть потоком подключей, в которых каждый подключ так или иначе зависит от позиции символа исходного текста, который используется для выбора подключа шифрования. Другими словами, мы должны иметь ключевой поток $k = (k_1, k_2, k_3 \ldots)$, в котором k_i применяется, чтобы зашифровать i -тый символ в исходном тексте и создать i -тый символ в зашифрованном тексте.

Автоключевой шифр

Чтобы понять зависимость ключа от позиции, обсудим простой

многоалфавитный шифр, названный "автоключевым". В этом шифре ключ — поток подключей, в котором каждый подключ используется, чтобы зашифровать соответствующий символ в исходном тексте. Первый подключ — определенное заранее значение, тайно согласованное Алисой и Бобом. Второй подключ — значение первого символа исходного текста (между 0 и 25). Третий — i -тое значение второго исходного текста. И так далее.

$P = P_1\ P_2\ P_3 \ldots..$
$C = C_1\ C_2\ C_3 \ldots..$
$K = (k_1, P_1, P_2, P_3, \ldots..)$
Шифрование $C_i = (P_i + k_i) \bmod 26$
Дешифрование $P_i = (C_i - k_i) \bmod 26$

Название шифра, автоключевой, подразумевает, что подключи создаются автоматически в зависимости от символов шифра исходного текста в процессе шифрования.

Пример 4.14

Предположим, что Алиса и Боб согласились использовать автоключевой шифр с начальным ключевым значением $k_1 = 12$. Теперь Алиса хочет передать Бобу сообщение "*Attack* is today" ("Атака — сегодня"). Шифрование проводится символ за символом. Каждый символ в исходном тексте сначала заменяется его значением целого числа, как показано на рис. 4.8, первый подключ прибавляется, чтобы создать первый символ зашифрованного текста. Остальная часть ключа создается по мере чтения символов исходного текста. Обратите внимание, что шифр является многоалфавитным, потому что эти три появления "a" в исходном тексте зашифрованы различно. Три возникновения "t" также зашифрованы различно.

Исходный текст	a	t	t	a	c	k	i	s	t	o	d	a	y
Значения P	00	19	19	00	02	10	08	18	19	14	03	00	24
Поток ключей	12	00	19	19	00	02	10	08	18	19	14	03	00
Значения C	12	19	12	19	02	12	18	00	11	07	17	03	24
Зашифрованный текст	M	T	M	T	C	M	S	A	L	H	R	D	Y

Автоключевой шифр действительно скрывает статистику частоты отдельного символа. Однако он так же уязвим при атаке с помощью грубой силы, как и аддитивный шифр. Первый подключ может быть только одним из 25 значений (1 – 25). Мы нуждаемся в многоалфавитных шифрах, которые не только скрывают характеристики языка, но и имеют большие множества ключей.

Шифр Плейфера

Другой пример многоалфавитного шифра — Шифр Плейфера, использовавшийся британской армией в течение Первой мировой войны. Ключ засекречивания в этом шифре сделан из 25 букв алфавита, размещенных в матрице 5×5 (буквы I и J рассматриваются при шифровании как одинаковые). С помощью различных соглашений о размещении букв в матрице можно создать много различных ключей засекречивания. Одно из возможных соглашений показано на рисунке 4.13.

Секретный ключ =

L	G	D	B	A
Q	M	H	E	C
U	R	N	I/J	F
X	V	S	O	K
Z	Y	W	T	P

Рис. 4.13. Пример секретного ключа Плейфера

Перед шифрованием исходный текст разбивается на пары; если две буквы пары одинаковые, то, чтобы отделить их, вставляется фиктивная буква. После вставки фиктивных букв, если число символов в исходном тексте нечетно, в конце добавляется один дополнительный фиктивный символ, чтобы сделать число символов четным.

Шифр использует три правила для шифрования:

а. если эти две буквы-пары расположены в одной и той же строке таблицы ключа засекречивания, соответствующий зашифрованный

символ для каждой буквы — следующий символ справа в той же самой строке (с возвращением к началу строки; если символ исходного текста — последний символ в строке);

b. если эти две буквы-пары расположены в одном и том же столбце таблицы ключа засекречивания, соответствующий зашифрованный символ для каждой буквы — символ ниже этого в том же самом столбце (с возвращением к началу столбца; если символ исходного текста — последний символ в столбце);

c. если эти две буквы-пары не находятся в одной строке или столбце таблицы засекречивания, соответствующий зашифрованный символ для каждой буквы — символ, который находится в его собственной строке, но в том же самом столбце, что и другой символ.

Шифр Плейфера соответствует нашим критериям для многоалфавитного шифра. Ключ — поток подключей, в котором они создаются по два одновременно. В шифре Плейфера поток ключей и поток шифра — те же самые. Это означает, что вышеупомянутые правила можно представить как правила для создания потока ключей. Алгоритм кодирования берет пару символов из исходного текста и создает пару подключей, следуя указанным правилам. Мы можем сказать, что поток ключей зависит от позиции символа в исходном тексте. Зависимость от позиции имеет здесь различную интерпретацию: подключ для каждого символа исходного текста зависит от следующего или предыдущего "соседа". Рассматривая шифр Плейфера, таким образом, можно сказать, что зашифрованный текст — это фактически поток ключей.

$P = P_1P_2P_3\ldots..$
$C = C_1C_2C_3\ldots..$
$k = [(k_1,k_2), (k_3,k_4),\ldots..]$
Шифрование $C_i = k_i$
Дешифрование $P_i = k_i$

Пример 4.15

Пусть нам надо зашифровать исходный текст `"hello"`, использующий ключи на рис. 4.13. Когда мы группируем буквы по парам, мы получаем `"he, ll,o"`. Мы должны вставить `x` между двумя `l` (эль), после

чего получим "he, lx, lo". Мы имеем

he -> EC lx -> QZ lo -> BX
Исходный текст: hello Зашифрованный текст: ECQZBX

Мы можем видеть из этого примера, что наш шифр — фактически многоалфавитный шифр: два появления l (эль) зашифрованы как "Q" и "B".

Криптоанализ шифра Плейфера

Очевидно, атака грубой силы шифра Плейфера очень трудна. Размер домена — 25! (факториал 25). Кроме того, шифровка скрывает частоту отдельных букв.

Однако частоты двухбуквенных комбинаций (диаграмм) сохранены (до некоторой степени из-за вставки наполнителя), так что криптоаналитик может использовать атаку только для зашифрованного текста, основанную на испытании частоты диаграмм, чтобы найти ключ.

Шифр Виженера

Один интересный вид многоалфавитного шифра был создан Блезом де Виженером, французским математиком шестнадцатого столетия. Шифр Виженера использует различную стратегию создания потока ключей. Поток ключей — повторение начального потока ключа засекречивания длины m, где мы имеем $1 < m < 26$. Шифр может быть описан следующим образом: $(k_1, k_2, \ldots, k_m)$ — первоначальный ключ засекречивания, согласованный Алисой и Бобом.

$P = P_1P_2P_3\ldots$
$C = C_1C_2C_3\ldots$
$k = [(k_1,k_2), (k_3,k_4),\ldots]$
Шифрование $C_i = k_i$
Дешифрование $P_i = k_i$

Одно важное отличие между шифром Виженера и другими двумя

многоалфавитными шифрами, которые мы рассмотрели: поток ключей Виженера не зависит от символов исходного текста; он зависит только от позиции символа в исходном тексте. Другими словами, поток ключей может быть создан без знания сути исходного текста.

Пример 4.16

Посмотрим, как мы можем зашифровать сообщение "She is listening (Она слушает) ", используя ключевое слово на 6 символов "PASCAL". Начальный поток ключей — это (15, 0, 18, 2, 0, 11). Поток ключей — повторение этого начального потока ключей (столько раз, сколько необходимо).

Исходный текст	s	h	e	i	s	l	i	s	t	e	n	i	n	g
Значения P	18	07	04	08	18	11	08	18	19	04	13	08	13	06
Поток ключей	15	00	18	02	00	11	15	00	18	02	00	11	15	00
Значения C	07	07	22	10	18	22	23	18	11	6	13	19	02	06
Шифрованный текст	H	H	W	K	S	W	X	S	L	G	N	T	C	G

Пример 4.17

Шифр Виженера может рассматриваться как комбинации аддитивных шифров. Рисунок 4.14 показывает, что исходный текст предыдущего примера можно рассматривать как состоящий из нескольких частей по шесть элементов в каждом (хотя в одном не хватило букв исходного текста), где каждый из элементов зашифрован отдельно. Рисунок поможет нам позже понять криптоанализ шифров Виженера. Имеется m частей исходного текста, каждый зашифрован различным ключом, чтобы разделить зашифрованный текст на m. частей.

Пример 4.18

Разобрав пример 4.18, мы убедимся, что аддитивный шифр — частный случай шифра Виженера, в котором m = 1.

Список Виженера

Другой способ рассмотрения шифров Виженера — с помощью того, что названо списком Виженера (Vigenere tableau) и показано в таблице 4.3.

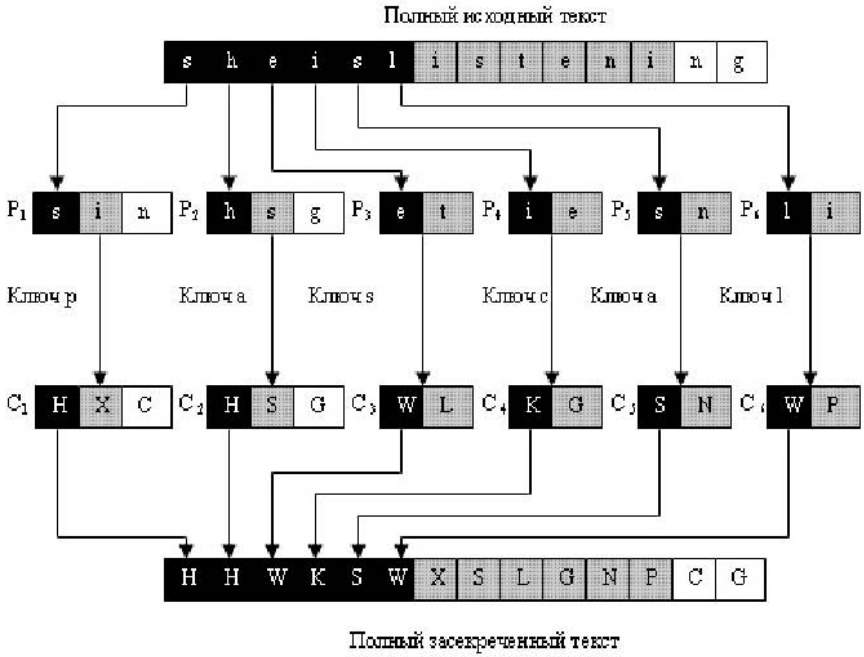


Рис. 4.14. Шифр Виженера как комбинация аддитивных шифров

Таблица 4.3. Список Виженера

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

L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X

Первая строка показывает символы исходного текста, который будет зашифрован. Первая колонка содержит столбец символов, которые используются ключом. Остальная часть таблицы показывает символы зашифрованного текста. Чтобы найти зашифрованный текст для исходного текста "she *listening*", используя слово "PASCAL" как ключ, мы можем найти "s" в первой строке, "P" в первом столбце, на пересечении строки и столбца — символ из зашифрованного текста "H". Находим "h" в первой строке и "A" во втором столбце, на пересечении строки и столбца — символ "H" из зашифрованного текста. И повторяем те же действия, пока все символы зашифрованного текста не будут найдены.

Криптоанализ шифра Виженера

Шифры Виженера, подобно всем многоалфавитным шифрам, не сохраняют частоту символов. Однако Ева может использовать некоторые методы для того, чтобы расшифровать перехваченный зашифрованный текст. Криптоанализ в данном случае состоит из двух

частей: находят длину ключа и потом непосредственно находят ключ.

1. Были изобретены несколько методов, чтобы найти длину ключа. Один метод рассмотрим ниже. В так называемом тесте Казиского (Kasiski) криптоаналитик в зашифрованном тексте ищет повторные сегменты по крайней мере из трех символов. Предположим, что найдены два сегмента, и расстояние между ними d. Криптоаналитик предполагает, что m делит d, где m — длина ключа. Если можно найти больше повторных сегментов с расстоянием d_1, d_2, …., d_n, тогда НОД(d_1, d_2, …., d_n....)/ m. Это предположение логично, потому что если два символа одинаковы и — $k \times m$ (k = 1, 2...) — символы, выделенные в исходном тексте, то одинаковы и $k \times m$ символы, выделенные в зашифрованном тексте. Криптоаналитик использует сегменты по крайней мере из трех символов, чтобы избежать случаев, где символы имеют один и тот же ключ. Пример 4.20 может помочь нам п онять эти рассуждения.
2. После того как длина ключа была найдена, криптоаналитик использует идею, показанную в примере 4.18. Здесь зашифрованный текст делится на m различных частей и применяется метод, используемый в криптоанализе аддитивного шифра, включая атаку частоты. Каждая часть зашифрованного текста может быть расшифрована и соединена с другими, чтобы создать целый исходный текст, другие слова. Весь зашифрованный текст не сохраняет частоту отдельной буквы исходного текста, но каждая часть делает это.

Пример 4.19

Предположим, что мы перехватили следующий зашифрованный текст:

```
LIOMWGFEGGDVWGHHCQUCRHRWAGWIOWQLKGZETKKMEVLWPCZVG'
VTSGXQOVGCSVETQLTJSUMVWVEUVLXEWSLGFZMVVWLGYHCUSWXQ
-
KVGSHEEVFLCFDGVSUMPHKIRZDMPHHBVWVWJWIXGFWLTSHGJOUE
VUCFVGOWICQLIJSUXGLW
```

Тест Казиского на повторение сегментов на три символа приводит к

результатам, показанным в таблице 4.4.

Таблица 4.4. Тест Казиского для примера 4.19

Комбинация	Первое расстояние	Второе расстояние	Разность
JSU	68	168	100
SUM	69	117	48
VWV	72	132	60
MPH	119	127	8

Наибольший делитель - 4, что означает длину ключа, пропорциональную 4. Сначала пробуем m = 4. Делим зашифрованный текст на четыре части. Часть C_1 состоит из символов 1, 5, 9...; часть C_2 состоит из символов 2, 6, 10..., и так далее. Используем статистическую атаку каждой части отдельно. Перебираем расшифровывающиеся части по одному символу одновременно, чтобы получить целый исходный текст.

C_1	LWGW	CRAO	KTEP	GTQC	TJVU	EGVG	UQGE	CVPR	PVJG	T
P_1	jueu	apym	ircn	eroa	rhts	thin	ytra	hcie	ixst	h
C_2	IGGG	QHGW	GKVC	TSOS	QSWV	WFVY	SHSV	FSHZ	HWWF	S
P_2	usss	ctsl	swho	feae	ceih	cete	soec	atnp	nkhe	r
C_3	OFDN	URWQ	ZKLZ	HGVV	LUVL	SZWH	WKHF	DUKD	HVIW	H
P_3	lcae	rotn	whiw	edss	irsi	irh	eteh	retl	tiid	e
C_4	MEVN	CWIL	EMWV	VXGE	TMEX	LMLC	XVEL	GMIM	BWHL	G
P_4	iard	yseh	aisr	rtca	piaf	pwte	thec	arha	esft	e

Если исходный текст не имеет смысла, попробуем с другим m.

В рассматриваемом случае исходный текст имеет смысл (читаем по столбцам).

Julius Caesar used a cryptosystem in his wars, which is now referred to as Caesar chipper. It is an additive chipper with the key set to three. Each character in the plaintext is shift three characters to create ciphertext.

Перевод этого текста приведен ниже.

Юлий Цезарь использовал в своих войнах криптографическую систему, которая упоминается теперь как шифр Цезаря. Это аддитивный шифр с ключом, установленным на три. Каждый символ в исходном тексте сдвинут на три символа, чтобы создать зашифрованный текст.

Шифр Хилла

Другой интересный пример многоалфавитного шифра — шифр Хилла, изобретенный Лестером С. Хиллом. В отличие от других многоалфавитных шифров, которые мы уже рассмотрели, здесь исходный текст разделен на блоки равного размера. Блоки зашифрованы по одному таким способом, что каждый символ в блоке вносит вклад в шифрование других символов в блоке. По этой причине шифр Хилла принадлежит к категории шифров, названных блочными шифрами. Другие шифры, которые мы изучали до сих пор, принадлежат к категории, называемой шифры потока. Отличие между шифрами блока и шифрами потока обсуждаются в конце этой лекции.

В шифре Хилла ключ — квадратная матрица размера $m \times m$., в котором m. является размером блока. Если мы вызываем ключевую матрицу K, то каждый элемент $k_{i,j}$ определяется матрицей, как показано на рис. 4.15.

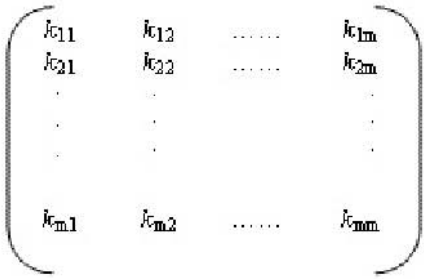


Рис. 4.15. Ключ в шифре Хилла

Покажем, как получается один блок зашифрованного текста. Если мы обозначим m символов блоков исходного текста P_1, P_2 ..., P_m, соответствующие символы в блоках зашифрованного текста будут C_1,

C_2 , ..., C_m. Тогда мы имеем

$$C_1 = P_1k_{11}+P_2k_{21}+\ldots\ldots+P_mk_{m1}$$
$$C_2 = P_1k_{12}+P_2k_{22}+\ldots\ldots+P_mk_{m2}$$
$$\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots$$
$$C_m = P_1k_{1m}+ P_2k_{2m} +\ldots\ldots+ P_mk_{mm}$$

Уравнения показывают, что каждый символ зашифрованного текста, такой, как C_1, зависит от символов всего исходного текста в блоке (P_1, P_2, ..., P_m). Однако мы должны знать, что не все квадратные матрицы имеют мультипликативные инверсии в Z_{26}, так что Алиса и Боб должны быть осторожны в выборе ключа. Боб не сможет расшифровать зашифрованный текст, передаваемый Алисой, если матрица не имеет мультипликативной инверсии.

Ключевая матрица в шифре Хилла должна иметь мультипликативную инверсию.

Пример 4.20

Использование матриц позволяет Алисе зашифровать весь исходный текст. В этом случае исходный текст — $l \times m$ — матрица, в которой l является номером блоков. Например, исходный текст "code is ready" ("код готов"), может быть представлен как матрица 3×4 при добавлении дополнительного фиктивного символа "z" к последнему блоку и удалении пробелов; зашифрованный текст выглядит как "OHKNIHGKLISS". Боб может расшифровать сообщение, используя инверсную матрицу-ключ. Шифрование и дешифрование показано на рис. 4.16.

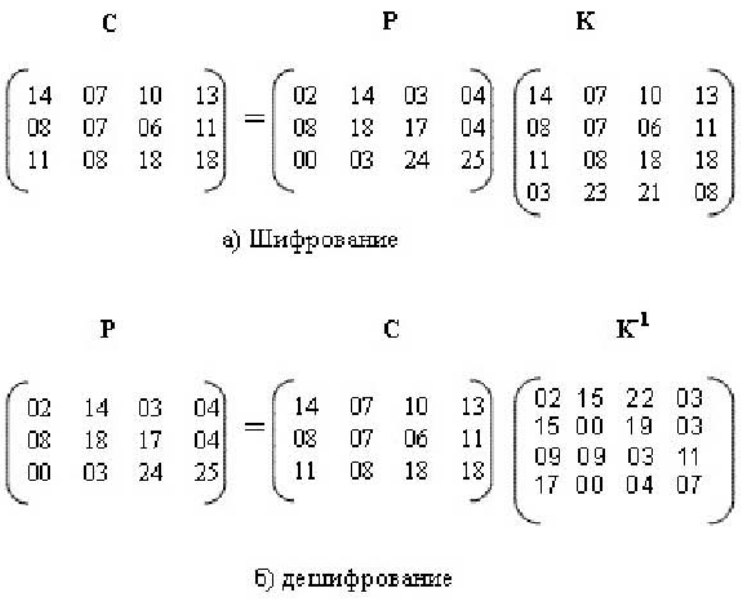


Рис. 4.16. Пример 4.20

Криптоанализ шифров Хилла

Криптоанализ только для зашифрованного шифрами Хилла текста труден. Во-первых, атака грубой силы при шифре Хилла чрезвычайно сложна, потому что матрица-ключ — $m \times m$. Каждый вход может иметь одно из 26 значений. Во-первых, это означает размер ключа $26^{m \times m}$. Однако, не все матрицы имеют мультипликативную инверсию. Поэтому область существования ключей все же не такая огромная.

Во-вторых, шифры Хилла не сохраняют статистику обычного текста. Ева не может провести анализ частоты отдельных букв из двух или трех букв. Анализ частоты слов размера m мог бы сработать, но очень редко исходный текст имеет много одинаковых строк размера m. Ева, однако, может провести атаку на шифр, используя метод знания исходного текста, если она знает значение m и знает пары "исходный текст/ зашифрованный текст", по крайней мере m блоков. Блоки могут принадлежать тому же самому сообщению или различным сообщениям,

но должны быть различны. Ева может создать две $m \times m$. матрицы, P (обычный текст) и C (зашифрованный текст), в котором соответствующие строки представляют известные пары обычного/ зашифрованного текста. Поскольку $C = PK$, Ева может использовать отношения $K = CP^{-1}$, чтобы найти ключ, если P является обратимым. Если P не является обратимым, то Ева должна задействовать различные наборы m пар обычного/зашифрованного текста.

Если Ева не знает значение m, она может попробовать различные значения при условии, что m не является очень большим.

Пример 4.21

Предположим, что Ева знает, что $m = 3$. Она перехватила три пары блока исходного/зашифрованного текста (не обязательно из того же самого сообщения), как показано на рис. 4.17.

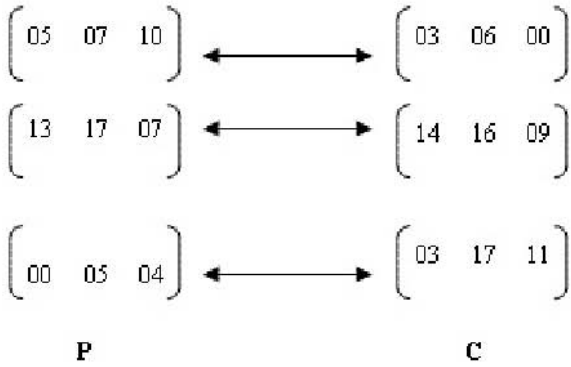


Рис. 4.17. Пример 4.22, формирования шифра зашифрованного текста

Она составляет матрицы P и C из этих пар. Поскольку в данном случае матрица P обратима, она инвертирует эту матрицу и умножает ее на C, что дает матрицу ключей K, как это показано на рис. 4.18.

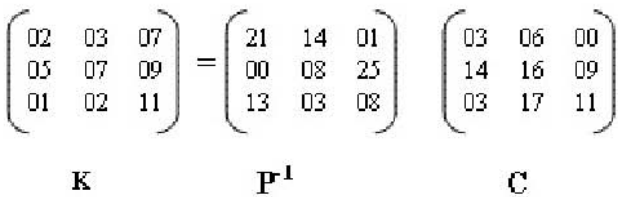


Рис. 4.18. Пример 4.22, поиска ключа

Теперь она имеет ключ и может взломать любой шифрованный текст, где применен этот ключ.

Одноразовый блокнот

Одна из целей криптографии — идеальная секретность. Исследования Шеннона показали, что идеальная секретность может быть достигнута, если символы исходного текста зашифрованы с помощью ключа, выбранного случайно из некоторой области ключей. Например, аддитивный шифр может быть легко взломан, потому что используется один и тот же ключ. Но даже и этот шифр может быть идеальным, если ключ, который применяется для шифрования каждого символа, выбран случайно из множества ключей (00, 01, 02.... 25): если первый символ зашифрован с помощью ключа 04, второй символ — с помощью ключа 02, третий — с помощью ключа 21, и так далее. Атака "только для зашифрованного текста" становится невозможна. Если передатчик изменяет ключ, используя каждый раз иную случайную последовательность целых чисел, другие типы атак также будут невозможны.

Эта идея используется в шифре, который называется одноразовым блокнотом. Его изобрел американский инженер Вернам. В этом шифре ключ имеет ту же самую длину, что и исходный текст, и выбран совершенно случайно.

Одноразовый блокнот — идеальный шифр, но его почти невозможно реализовать коммерчески. Если ключ каждый раз генерируется заново, как Алиса может каждый раз сообщать Бобу новый ключ? Для этого каждый раз нужно передавать сообщение. Однако есть некоторые случаи, когда возможно использование одноразового блокнота. Например, если президент страны должен передать полностью секретное сообщение президенту другой страны, он может перед посылкой сообщения передать с помощью доверенного посланника случайный ключ. Некоторые вопросы изменения шифра одноразового блокнота обсуждаются в дальнейших лекциях, когда будет рассматриваться введение в современную криптографию.

Роторный шифр

Хотя шифры одноразового блокнота не применяются на практике, один шаг от него к более защищенному шифру — роторный шифр. Он возвращается к идее моноалфавитной подстановки, но меняет принцип отображения исходного текста в символы зашифрованного текста для каждого символа исходного текста. Рисунок 4.19 показывает упрощенный пример роторного шифра.

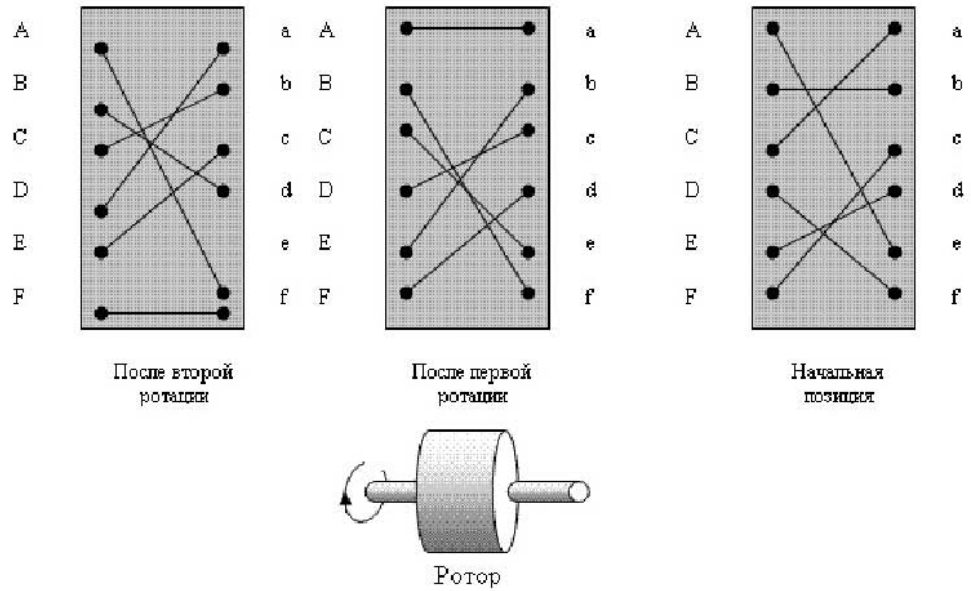


Рис. 4.19. Роторный шифр

Ротор, показанный на рис. 4.19, применен только для 6 букв, но реальные роторы используют 26 букв. Ротор постоянно связывает символы исходного и зашифрованного текстов, но подключение обеспечивается щетками. Обратите внимание, что соединение символов исходного и зашифрованного текстов показано так, как если бы ротор был прозрачен и можно было видеть внутреннюю часть.

Начальная установка (позиция) ротора — ключ засекречивания между Алисой и Бобом — это зашифрованный первый символ исходного текста. Используя начальную установку, второй символ зашифрован после того, как проведено первое вращение (на рис. 4.19 — это поворот

на 1/6 круга, на реальной установке — поворот на 1/26), и так далее.

Слово с тремя буквами, такими как "bee", зашифровано как "*BAA*", если ротор неподвижен (моноалфавитный шифр подстановки), но оно будет зашифровано как "*BCA*", если он вращается (роторный шифр). Это показывает, что роторный шифр — многоалфавитный шифр, потому что два появления того же самого символа исходного текста зашифрованы как различные символы.

Роторный шифр является стойким к атаке грубой силы, как моноалфавитный шифр подстановки, потому что Ева должна найти первое множество отображений среди возможных 26! (факториал). Роторный шифр является намного более стойким к статистической атаке, чем моноалфавитный шифр подстановки, потому что в нем не сохраняется частота употребления буквы.

Машина "Энигма"

Машина "Энигма" была первоначально изобретена в Сербии, но была изменена специалистами немецкой армии и интенсивно использовалась в течение Второй Мировой Войны. Машина базировалась на принципе шифров ротора. Рисунок 4.20 показывает упрощенную схему построения машины.

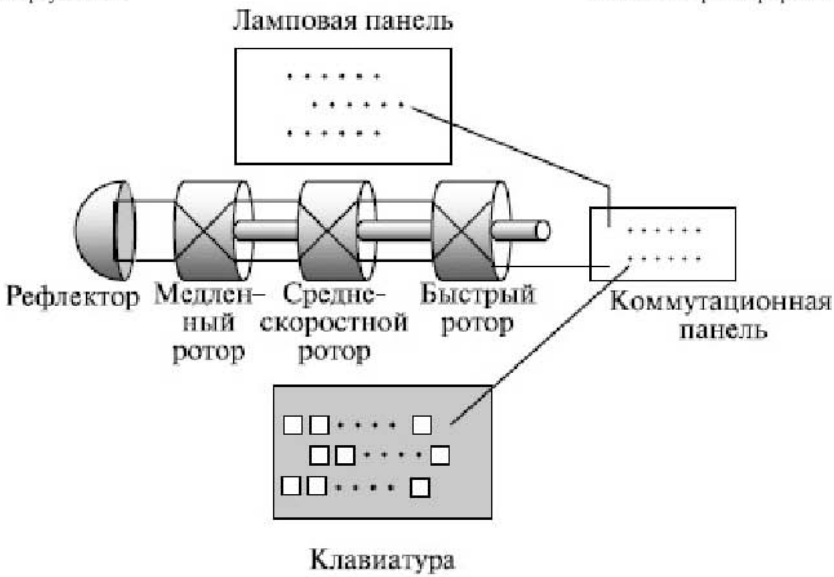


Рис. 4.20. Примерное построение Машины Энгима

Ниже перечислены главные компоненты машины.

1. Клавиатура с 26 -ю ключами, используемыми для того, чтобы вводить исходный текст при шифровании, и для того, чтобы вводить зашифрованный текст при расшифровке.
2. Ламповая панель с 26 -ю лампами, которая показывает символы зашифрованного текста при шифровании и символы исходного текста при дешифровании.
3. Коммутационная панель с 26 -ю штепселями, вручную подключенными 13 -ю проводами. Конфигурация изменяется каждый день, чтобы обеспечить различное скрэмблирование.
4. Три замонтированных ротора, такие же как рассмотренные в предыдущей секции. Эти три ротора выбираются ежедневно из пяти доступных роторов. Быстрый ротор вращается на 1/26 поворота при каждом символе, введенном с помощью клавиатуры. Средний ротор делает 1/26 поворота при каждом полном повороте быстрого ротора. Медленный ротор делает 1/26 поворота для каждого законченного поворота среднего ротора.
5. Отражатель, который является постоянным и предварительно замонтированным.

Кодовая книга — справочник шифров

Чтобы использовать "Энигму", была издана кодовая книга, которая в течение каждого дня дает несколько параметров настройки, включая:

a. три ротора, которые должны быть выбраны из пяти доступных;

b. порядок, в котором эти роторы должны быть установлены;

c. параметры установок для коммутационной панели;

d. код с тремя буквами дня.

Процедура шифровавшая Сообщения

Чтобы зашифровать сообщение, оператор должен последовательно сделать шаги, перечисленные ниже:

1. установить стартовую позицию роторов согласно коду дня. Например, если код был `"HUA"`, роторы должны быть инициализированы на `"H"`, `"U"` и `"A"` соответственно;
2. выбрать случайный код с тремя буквами, например *ACF*. Зашифровать текст `"ACFACF"` (повторный код), используя начальную установку роторов шага `1`. Например, предположим, что зашифрованный код — `"OPNABT"` ;
3. установить стартовые позиции роторов к `OPN` (половина зашифрованного кода);
4. добавить зашифрованные шесть букв, полученных на шаге `2` (`"OPNABT"`), в конец к начальному сообщению;
5. зашифровать сообщение, включая код с `6` -ю буквами. Передать зашифрованное сообщение.

Процедура для расшифровки сообщения

Чтобы расшифровывать сообщение, оператор должен сделать следующие шаги:

1. получить сообщение и отделить первые шесть букв;
2. установить стартовую позицию роторов согласно коду дня;
3. расшифровать первые шесть букв писем, используя начальную установку шага 2 ;
4. установить позиции роторов на первую половину расшифрованного кода;
5. расшифровать сообщение (без первых шести букв).

Криптоанализ

Мы знаем, что "Энигма" во время войны была взломана, хотя германская армия и остальная часть мира не знала об этом факте еще несколько десятилетий после того. Вопрос: как такой сложный шифр был атакован? Хотя немецкий язык весьма сложен, союзники, так или иначе, получили некоторые копии машин. Следующим шагом был поиск параметров установки в течение каждого дня и кода, передаваемого для инициализации роторов для каждого сообщения. Изобретение первого компьютера помогло союзникам преодолеть эти трудности. Подробное изображение машины "Энигма" и ее криптоанализ может быть найден на сайтах, посвященных этой машине.

4.3. Шифры перестановки

Шифр перестановки не заменяет одним символом другой, вместо этого он изменяет местоположение символов. Символ в первой позиции исходного текста может появиться в десятой позиции зашифрованного текста. Символ, который находится в восьмой позиции исходного текста, может появиться в первой позиции зашифрованного текста. Другими словами, шифр перестановки ставит в другом порядке (перемещает) символы.

Шифр перестановки меняет порядок следования символов.

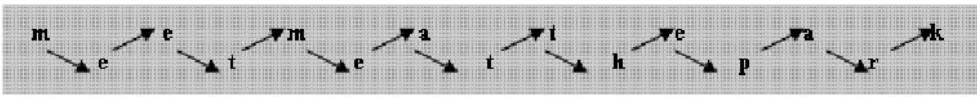
Шифры перестановки без использования ключа

Простые шифры перестановки, которые применялись в прошлом, не

использовали ключ. Есть два метода для перестановки символов. В первом методе текст записывается в таблице столбец за столбцом и затем передаётся строка за строкой. Во втором методе текст написан в таблицы строка за строкой и затем передаётся столбец за столбцом.

Пример 4.22

Хороший пример шифра без использования ключа — шифр изгороди (rail fence cipher). В этом шифре исходный текст размещен на двух линиях как зигзагообразный шаблон (что может рассматриваться как столбец за столбцом таблицы, которая имеет две строки); зашифрованный текст составляется при чтении шаблона строка за строкой. Например, чтобы передать сообщение "Meet me at the *park*" ("Встречай меня в парке"), Алиса пишет Бобу:



Алиса создает зашифрованный текст "MEMATEAKETETHPR", посылая первую строку, сопровождаемую второй строкой. Боб получает зашифрованный текст и разделяет его пополам (в этом случае вторая половина имеет на один символ меньше). Первая половина формы — первая строка; вторая половина — вторая строка. Боб читает результат по зигзагу. Поскольку нет никакого ключа и номер строк установлен (2), криптоанализ зашифрованного текста был бы очень прост для Евы. Все, что она должна знать, — это то, что используется шифр изгороди.

Пример 4.23

Алиса и Боб могут договориться о числе столбцов и использовать второй метод. Алиса пишет тот же самый исходный текст, строка за строкой, в таблице из четырех столбцов.

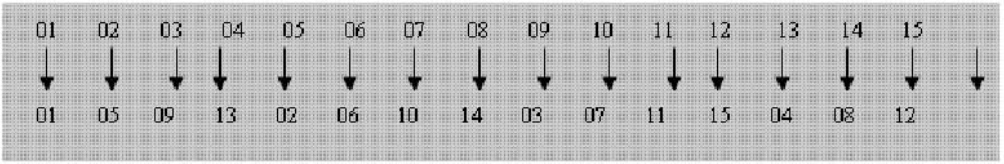
m	e	e	t
m	e	a	t
t	h	e	p
a	r	k	

Алиса создает зашифрованный текст "MMTAEEHREAEK *TTP*",

передавая символы столбец за столбцом. Боб получает зашифрованный текст и применяет обратный процесс. Он пишет полученное сообщение столбец за столбцом и читает его строка за строкой как исходный текст. Ева может легко расшифровать сообщение, если она знает число столбцов.

Пример 4.24

Шифр в примере 4.23 — реальный шифр перестановки. Далее покажем перестановку каждой буквы исходного текста и зашифрованный текст, базируясь на номерах их позиций.



Второй символ в исходном тексте передвинулся на пятую позицию в зашифрованном тексте; третий символ передвинулся на девятую позицию; и так далее. Хотя символы переставлены, они сами являются шаблонами: `(01, 05, 09, 13)`, `(02, 06, 10, 14)`, `(03, 07, 11, 15)` и `(04, 08, 12)`. В каждой секции разность между двумя смежными номерами — `4`.

Ключевые шифры перестановки

Бесключевые шифры переставляют символы, используя запись исходного текста одним способом (например, строка за строкой) и передачу этого текста в другом порядке (например, столбец за столбцом). Перестановка делается во всём исходном тексте, чтобы создать весь зашифрованный текст. Другой метод состоит в том, чтобы разделить исходный текст на группы заранее определенного размера, называемые блоками, а затем использовать ключ, чтобы переставить символы в каждом блоке отдельно.

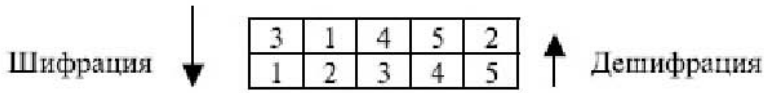
Пример 4.25

Алиса должна передать Бобу сообщение "`Enemy attacks`

tonight" ("Вражеские атаки сегодня вечером"). Алиса и Боб согласились разделить текст на группы по пять символов и затем переставить символы в каждой группе. Ниже показана группировка после добавления фиктивного символа в конце, чтобы сделать последнюю группу одинаковой по размеру с другими.

Enemy atttac kston ightz

Ключ, используемый для шифрования и дешифрования, — ключ перестановки, который показывает, как переставлять символы. Для этого сообщения примем, что Алиса и Боб использовали следующий ключ:



Третий символ в блоке исходного текста становится первым символом в зашифрованном тексте в блоке, первый символ в блоке исходного текста становится вторым символом в блоке зашифрованного текста и так далее. Результаты перестановки:

EEMYN TAACT TKONS HITZG

Алиса передает зашифрованный текст "EEMYNTAACTTKONSHITZG" Бобу. Боб делит зашифрованный текст на группы по 5 символов и, используя ключ в обратном порядке, находит исходный текст.

Объединение двух подходов

Современные шифры перестановки, чтобы достигнуть лучшего скремблирования, объединяют два подхода. Шифрование и дешифрование делается в три шага. Первый: текст пишется таблицей строка за строкой. Второй: делается перестановка, изменяя порядок следования столбцов. Третий: столбец за столбцом читается новая таблица. Первые и третьи шаги обеспечивают бесключевое глобальное изменение порядка следования; второй шаг обеспечивает блочную ключевую перестановку. Эти типы шифров упоминаются часто как ключевые шифры перестановки столбцов.

Пример 4.26

Предположим, что Алиса снова зашифровывает сообщение в примере 4.25, на сей раз используя объединенный подход. Шифрование и дешифрование показано на рис. 4.21.

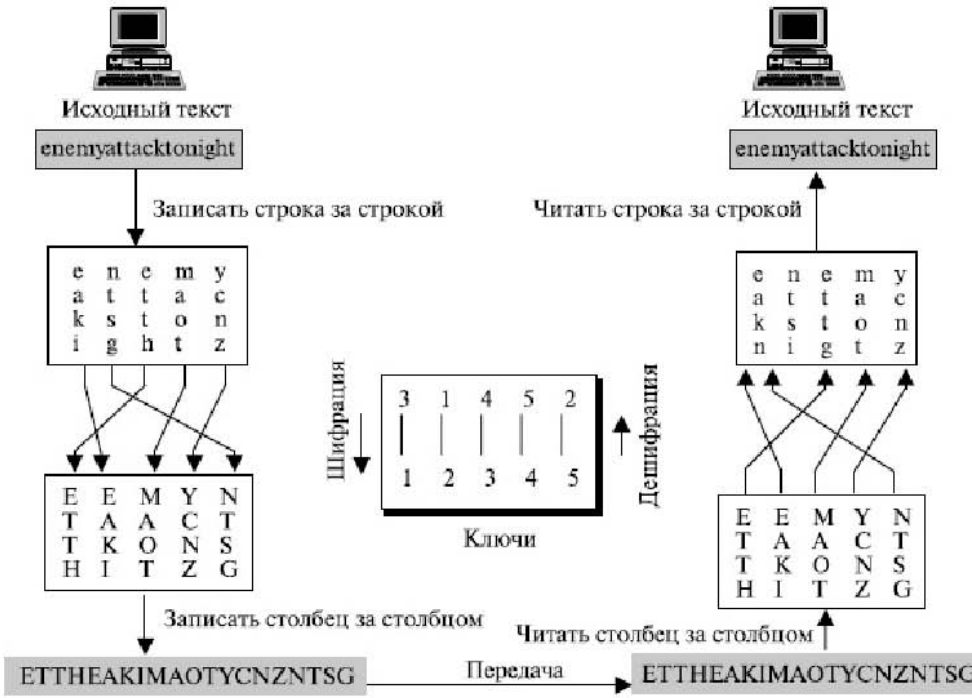


Рис. 4.21. Пример 4.27

Первая таблица, созданная Алисой, содержит исходный текст, записанный строка за строкой. Столбцы переставлены с использованием того же самого ключа, что и в предыдущем примере. Зашифрованный текст создан с помощью чтения второй таблицы столбец за столбцом. Боб делает те же самые три шага в обратном порядке. Он считывает таблицу зашифрованного текста столбец за столбцом в первую таблицу, переставляет столбцы, а затем читает вторую таблицу строку за строкой.

Ключи

В примере 4.27 единственный ключ использовался в двух направлениях для изменения порядка следования столбцов — вниз для шифрования, вверх для дешифрования. Обычно принято создавать два ключа для этого графического представления: один для шифрования и один для дешифрования. Ключи накапливаются в таблицах, имеющих один адрес (вход) для каждого столбца. Вход содержит исходный номер столбца — номер столбца пункта назначения, указывающий его положение от номера входа. Рисунок 4.22 показывает, как эти две таблицы могут быть созданы с помощью графического представления ключа.

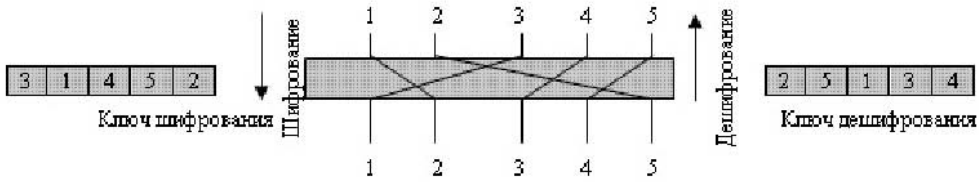
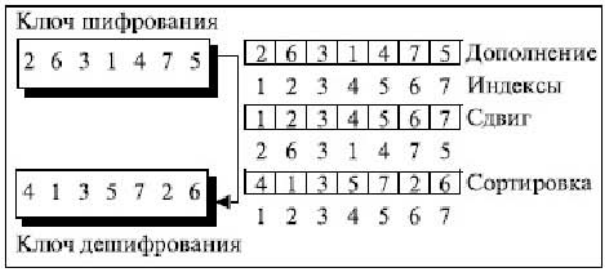


Рис. 4.22. Шифрование / дешифрование в шифре перестановки

Ключ шифрования — `(3 1 4 5 2)`. Первый вход показывает, что столбец `3` (содержание) в источнике становится столбцом `1` (положение или индекс входа) в пункте назначения. Ключ дешифрования — `(2 5 1 3 4)`. Первый вход показывает, что столбец `2` в источнике становится столбцом `1` в пункте назначения.

Как найти ключ дешифрования, если дан ключ шифрования или, наоборот, дан ключ дешифрации? Процесс может быть выполнен вручную за несколько шагов, как это показано на рис. 4.23. Сначала добавим индексы к таблице ключей, потом сделаем сдвиг в соответствии с полученным ключом, и, наконец, сортируем пару согласно индексу.



а. Ручной процесс

```
Given: EncKey {index}
index ← 1
while {index ⩽ Column}
 {
   DecKey [EncKey [index]←index
   Index ← index + 1
 }
Return: Dec Key [index]
```

б. Алгоритм

Рис. 4.23. Инверсия ключа в шифре перестановки

Использование матриц

Мы можем использовать матрицы, чтобы показать процесс шифрования/дешифрования для шифра перестановки. Исходный текст и зашифрованный текст — матрица $l \times m$., представляющая числовые значения символов; ключи — квадратные матрицы размера $m \times m$. В матрице перестановки каждая строка или столбец имеют строго одну единицу (1), и остальная часть значений — нули (0). Шифрование выполняется умножением матрицы исходного текста на ключевую матрицу, чтобы получить матрицу зашифрованного текста; дешифрование выполняется умножением зашифрованного текста на инверсию ключевой матрицы, после чего получаем исходный текст.

Очень интересно, что матрица дешифрования в этом случае, как и всегда, — инверсия матрицы шифрования. Однако нет никакой необходимости инвертировать матрицу — ключевая матрица шифрования может просто быть переставлена (сдвиг строк и столбцов), чтобы получить ключевую матрицу дешифрования.

Пример 4.27

Рисунок 4.24 показывает процесс шифрования. Умножение матрицы 4×5 исходного текста на ключевую матрицу шифрования 5×5 дает матрицу зашифрованного текста 4×5. Матричная манипуляция требует изменения символов в примере 4.27 к их числовым значениям (от 00 до 25). Обратите внимание, что матричное умножение

обеспечивает только перестановку столбцов; чтение и запись в матрицу должны быть обеспечены остальной частью алгоритма.

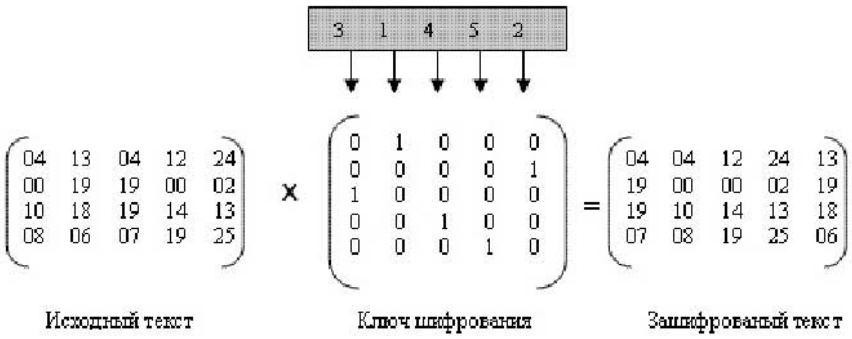


Рис. 4.24. Представление ключа в виде матрицы в шифре перестановок

Криптоанализ шифров перестановки

Шифры перестановки уязвимы к нескольким видам атак только для зашифрованного текста.

Статистическая атака

Шифр перестановки не изменяет частоту букв в зашифрованном тексте; он только переставляет буквы. Так что первая атака, которая может быть применена, - анализ частоты отдельной буквы. Этот метод может быть полезен, если длина зашифрованного текста достаточно большая. Мы такую атаку рассматривали раньше. Однако шифры перестановки не сохраняют частоту пар и триграмм. Это означает, что Ева не может использовать такие инструментальные средства. Фактически, если шифр не сохраняет частоту пар и триграмм, но сохраняет частоту отдельных букв, то вероятнее всего, что это шифр перестановки.

Атака грубой силы

Ева, чтобы расшифровать сообщение, может попробовать все возможные ключи. Однако число ключей может быть огромно `1! +`

2! + 3! + ... + L!, где L — длина зашифрованного текста. Лучший подход состоит в том, чтобы попробовать отгадать число столбцов. Ева знает, что число столбцов делится на L. Например, если длина шифра — 20 символов, то $20 = 1 \times 2 \times 2 \times 5$. Это означает, что номером столбцов может быть комбинация этих коэффициентов (1, 2, 4, 5, 10, 20). Однако только один столбец и только одна строка — маловероятные варианты.

Пример 4.28

Предположим, что Ева перехватила сообщение зашифрованного текста "EEMYNTAACTTKONSHITZG". Длина сообщения L = 20, число столбцов может быть 1, 2, 4, 5, 10 или 20. Ева игнорирует первое значение, потому что это означает только один столбец и оно маловероятно.

a. Если число столбцов — 2, единственные две перестановки — (1,2) и (2, 1). Первое означает, что перестановки не было. Ева пробует вторую комбинацию. Она делит зашифрованный текст на модули по два символа "EE MY NT AA CT TK ON SH IT ZG". Затем она пробует переставлять каждый модуль из них, получая текст "ee ym nt aa tc kt no hs ti gz", который не имеет смысла.

b. Если номер столбцов — 4, тогда имеется 4! = 24 перестановки. Первая перестановка (1 2 3 4) означает, что не было никакой перестановки. Ева должна попробовать остальные. После испытания всех 23 возможностей Ева находит, что никакой исходный текст при таких перестановках не имеет смысла.

c. Если число столбцов — 5, тогда есть 5! = 120 перестановок. Первая (1 2 3 4 5) означает отсутствие перестановки. Ева должна попробовать остальные. Перестановка (2 5 13 4) приносит плоды — исходный текст "enemyattackstonightz", который имеет смысл после удаления фиктивной буквы z и добавления пробелов.

Атака по образцу

Другая атака шифра перестановки может быть названа атакой по образцу. Зашифрованный текст, созданный с помощью ключевого шифра перестановки, имеет некоторые повторяющиеся образцы. Следующий пример показывает зашифрованный текст, относительно которого известно, что каждый символ в зашифрованном тексте в примере 4.28 получается из исходного текста по следующему правилу:

03 08 13 18 *01 06 11 16* **04 09 14 19** *05 10 15 20* **02 07 12 17**

1 -й символ в зашифрованном тексте получается из 3 -го символа исходного текста. 2 -й символ в зашифрованном тексте получается из 8 -го символа исходного текста. 20 -й символ в зашифрованном тексте получается из 17 -го символа исходного текста, и так далее. У нас имеются образцы в вышеупомянутом списке. Мы имеем пять групп: (3, 8, 13, 18), (1, 6, 11, 16), (4, 9, 14, 19), (5, 10, 15, 20) и (2, 7, 12, 17). Во всех группах разность между двумя смежными номерами — 5. Эта регулярность может использоваться криптоаналитиком, чтобы взломать шифр. Если Ева знает или может предположить число столбцов (в этом случае оно равняется 5), она может преобразовать зашифрованный текст в группы по четыре символа. Перестановка групп может обеспечить ключ к нахождению исходного текста.

Шифры с двойной перестановкой

Шифры с двойной перестановкой могут затруднить работу криптоаналитика. Примером такого шифра было бы повторение дважды алгоритма, используемого для шифрования и дешифрования в примере 4.26. На каждом шаге может применяться различный ключ, но обычно ключ используется один и тот же.

Пример 4.29

Повторим пример 4.26, где использована двойная перестановка. Рисунок 4.25 показывает процесс.

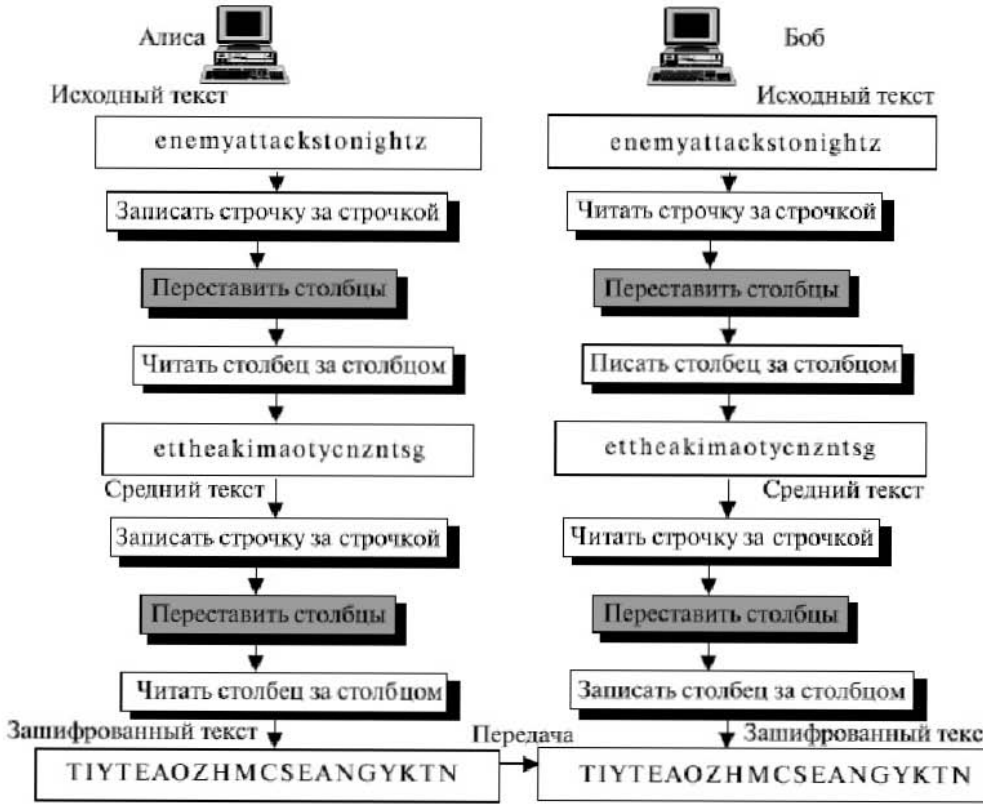


Рис. 4.25. Двойной шифр перестановки

Хотя криптоаналитик может еще использовать частоту появления отдельного символа для статистической атаки на зашифрованный текст, атака по образцу теперь затруднена.

13 16 05 07 03 06 10 20 18 04 10 12 01 09 15 17

Сравнив приведенный текст и результат примера 4.28, мы видим, что теперь нет повторяющихся образцов. Двойная перестановка удалила ту регулярность, что мы имели раньше.

4.4. Шифры потока и блочные шифры

В литературе симметричные шифры разделяют на две категории: шифры потока и блочные шифры. Хотя эти категории применяются к современным шифрам, но они могут также работать в традиционных

шифрах.

Шифры потока

В шифрах потока шифрование делается в один момент времени над одним символом (таким, как буква или бит). Мы имеем поток исходного текста, поток зашифрованного текста и поток ключей. Обозначим исходный поток P, поток зашифрованного текста — C и поток ключей — K.

$P = P_1P_2P_3,\ldots$
$C = C_1C_2C_3,\ldots$
$K = (k_1k_2k_3,\ldots)$
$C_1 = E_{k1}(P_1)$
$C_2 = E_{k2}(P_2)$
$C_3 = E_{k3}(P_3)\ldots$

Рисунок 4.26 показывает идею указанного ранее шифра потока. Символы обычного текста принимаются алгоритмом шифрования по одному. Символы зашифрованного текста также создаются по одному в один и тот же момент времени. Ключевой поток может быть создан многими способами. Это может быть поток с заранее определенными значениями; это может быть только одно значение, используемое алгоритмом. Значения могут зависеть от исходного текста или символов зашифрованного текста. Значения могут также зависеть от предыдущих ключевых значений.

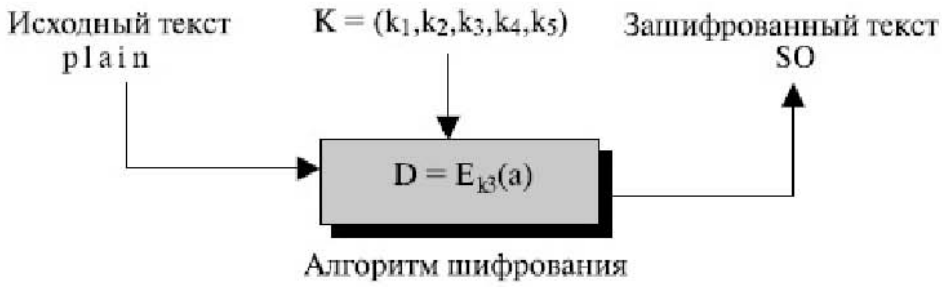


Рис. 4.26. Шифр потока

Рисунок 4.26 показывает момент, когда третий символ в потоке исходного текста был зашифрован с использованием третьего значения в ключевом потоке. Результат — это третий символ в потоке зашифрованного текста.

Пример 4.30

Аддитивные шифры могут быть отнесены к категории шифров потока, в которых ключевой поток является повторным значением ключа. Другими словами, ключевой поток рассматривают как заранее определенный поток ключей или K = (k, k, …k). В этом шифре, однако, каждый символ в зашифрованном тексте зависит только от соответствующего символа в исходном тексте, потому что ключевой поток генерировался независимо.

Пример 4.31

Моноалфавитные шифры подстановки, которые мы рассмотрели в этой лекции, — также шифры потока. Однако каждое значение ключевого потока в этом случае — отображение текущих исходных букв в соответствующие символы зашифрованного текста по таблице отображения.

Пример 4.32

Шифры Виженера — также шифры потока согласно определению. В этом случае ключ потока — повторение m значений, где m. — размер ключевого слова. Другими словами,

$$K = (k_1, k_2, \ldots k_m, k_1, k_2, \ldots\ldots k_m \ldots)$$

Пример 4.33

Мы можем установить критерий для разделения шифров потока, основанных на ключевых потоках. Мы можем сказать, что шифр потока — моноалфавитный шифр, если значение k_i не зависит от исходного символа исходного текста в потоке исходного текста; в противном случае шифр является многоалфавитным.

- Аддитивные шифры являются моноалфавитными, потому что k_i в ключевом потоке — зафиксированный (постоянный), он не зависит от позиции символа в исходном тексте.
- Моноалфавитные шифры подстановки являются явно моноалфавитными шифрами потока, потому что k_i не зависит от позиции соответствующего символа в потоке исходного текста, а зависит лишь от значения символа в исходном тексте.
- Шифры Виженера — многоалфавитные шифры, потому что k_i явно зависит от позиции символа исходного текста. Однако зависимость является циклической. Одинаковый ключ для двух символов разделен m позициями.

Блочные шифры

В блочном шифре группа символов исходного текста размера m $(m>1)$ зашифровывается, создавая вместе группу зашифрованного текста одного и того же размера. Основанный на этом определении блочный шифр использует единственный ключ, чтобы зашифровать целый блок, даже если ключ делает перемножение значений. Рисунок 4.27 показывает принципы блочного шифра.

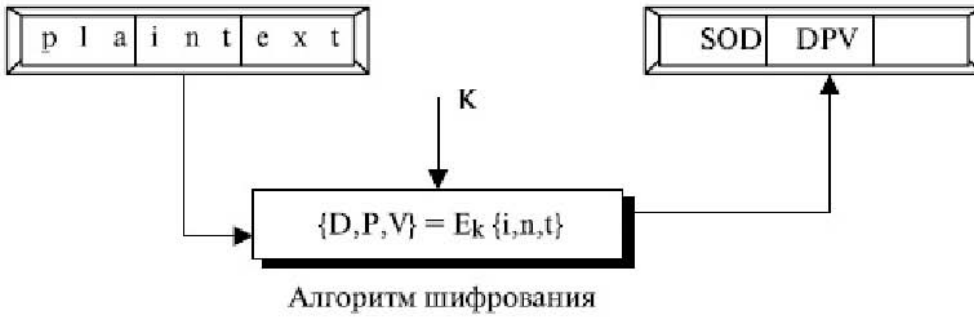


Рис. 4.27. Блочный шифр

В блочном шифре блок зашифрованного текста зависит от целого блока исходного текста.

Пример 4.34

Шифры Плейфера — блочные шифры. Размер блока — m = 2. Два символа зашифрованы вместе.

Пример 4.35

Шифры Хилла — блочные шифры. Блок исходного текста размера 2 или больше зашифрован, совместно используя единственный ключ (матрицу). В этих шифрах значение каждого символа в зашифрованном тексте зависит от всех значений символов в исходном тексте. Хотя ключ может быть получен из $m \times m$. значений, он рассматривается как единственный ключ.

Пример. 4.36

Из определения блочного шифра ясно, что каждый блочный шифр — это многоалфавитный шифр, потому что каждая буква шифрованного текста в случае блочного шифра зависит от всех букв исходного текста.

Комбинация

На практике блоки исходного текста шифруются индивидуально, но они используют ключи потока для того, чтобы зашифровать все сообщение блок за блоком. Другими словами, шифр — блочный, когда применяется к индивидуальным блокам, но он же и шифр потока, когда применяется ко всему сообщению, рассматривая каждый блок как единицу. Каждый блок использует различный ключ, который был сгенерирован заранее или в течение процесса шифрования. Примеры этого будут рассмотрены позднее.

4.5. Рекомендованная литература

Для более детального изучения положений, обсужденных в этой лекции, мы рекомендуем нижеследующие книги и сайты. Пункты, указанные в скобках, приведены в списке ссылок в конце книги.

Книги

Несколько книг рассматривают классические шифры с симметричным ключом. [Kah96] и [Sin99] дают полную историю этих шифров. [Sti06], [Bar02], [Cou99], [Sta06], [Mao04] и [Garol] приводят хороший анализ технических деталей.

Сайты

Нижеследующие сайты дают больше информации о темах, рассмотренных в этой лекции.

- ссылка: http://www.cryptogram.org - http://www.cryptogram.org
- ссылка: http://www.cdt.org/crypto/ - http://www.cdt.org/crypto/
- ссылка: http://www.cacr.math.uwaterloo.ca/ - http://www.cacr.math.uwaterloo.ca/
- ссылка: http://www.acc.stevens.edu/crypto.php - http://www.acc.stevens.edu/crypto.php
- ссылка: http://www.crypto .com/ - http://www.crypto.com/
- ссылка: http: // theory.lcs.mit.edu / ~ rivest/crypto-security.html - http://theory.lcs.mit.edu/~rivest/crypto-security.html
- ссылка: http://www.trincoll.edu/depts/cpsc/cryptography/substitution.html - http://www.trincoll.edu/depts/cpsc/cryptography/substitution.html
- ссылка: http://hem.passagen.se/tan01/transpo.html - http://hem.passagen.se/tan01/transpo.html
- ссылка: http://www.strangehorizons.comy2001/20011008/steganography.shtml - http://www.strangehorizons.comy2001/20011008/steganography.shtml

4.6. Итоги

- Шифрование симметричными ключами использует единственный ключ для шифрования и для дешифрования. Алгоритмы шифрования и дешифрования являются обратными друг друга.
- Первоначальное сообщение называется исходным текстом. Сообщение, которое передаётся через канал, называется зашифрованным текстом. Чтобы создавать зашифрованный текст из исходного текста, алгоритм шифрования пользуется общедоступным ключом засекречивания. Чтобы создавать

исходный текст из зашифрованного текста, используется алгоритм дешифрования тот же самый ключ засекречивания.

- На основании принципа Керкгоффса, нужно всегда принимать, что противник знает алгоритм шифрования/дешифрования. Устойчивость шифра к атаке должна быть основана только на тайне ключа.
- Криптоанализ — наука и искусство "взлома" шифров. Есть четыре общих типа атак криптоанализа: атака только на зашифрованный текст, атака по образцу, атака с выборкой исходного текста, атака с выборкой зашифрованного текста.
- Традиционные шифры с симметричным ключом могут быть разделены на две обширных категории: шифры подстановки и шифры перестановки. Шифр подстановки заменяет один символ другим символом. Шифр перестановки переупорядочивает символы.
- Шифры подстановки могут быть разделены на две обширных категории: моноалфавитные шифры и многоалфавитные шифры. В моноалфавитной подстановке отношения между символом в исходном тексте и символом в зашифрованном тексте являются непосредственными. В многоалфавитной подстановке отношения между символами в исходном тексте и символами в зашифрованном тексте — "один ко многим".
- Моноалфавитные шифры включают в себя аддитивные, мультипликативные, аффинные и моноалфавитные шифры подстановки.
- Многоалфавитные шифры включают в себя шифры: автоключевой, Плейфера, Виженера, Хилла, одноразового блокнота, ротора, и шифры "Энигмы".
- Шифры перестановки включают в себя: бесключевой, ключевой шифры и шифры с двойной перестановкой.
- Симметричные шифры могут также быть разделены на две обширных категории: шифры потока и блочные шифры. В шифре потока шифрование и дешифрование одного символа производятся в один момент времени. В блочном шифре символы в блоке зашифрованы вместе. Практически, блоки исходного текста зашифрованы индивидуально, но они используют поток ключей, чтобы зашифровать все сообщение блок за блоком.

4.7. Набор для практики

Обзорные вопросы

1. Определите шифр с симметричным ключом.
2. Поясните отличия между шифром подстановки и шифром перестановки.
3. Поясните отличия между моноалфавитным и многоалфавитным шифрами.
4. Поясните отличия между шифром потока и блочным шифром.
5. Все ли шифры потока являются моноалфавитными? Поясните.
6. Все ли блочные шифры являются многоалфавитными? Поясните.
7. Перечислите три моноалфавитных шифра.
8. Перечислите три многоалфавитных шифра.
9. Перечислите два шифра перестановки.
10. Перечислите четыре вида атак криптоанализа.

Упражнения

1. Маленький частный клуб имеет только `100` членов. Ответьте на следующие вопросы:
    - Сколько ключей засекречивания необходимо иметь, если все члены клуба хотят передавать секретные сообщения друг другу?
    - Сколько ключей засекречивания необходимо, если каждый доверяет президенту клуба? Если один член клуба должен передать сообщение другому, он сначала передает это президенту; президент тогда передает сообщение другому члену клуба.
    - Сколько ключей засекречивания необходимо, если президент решает, что два члена клуба, которые должны связаться друг с другом, должны сначала войти в контакт с ним? Президент тогда создает временный ключ, который используется между этими двумя членами клуба. Временный ключ зашифровывается и посылается обоим членам клуба.
2. Археологи нашли новый манускрипт, написанный на

неизвестном языке. Позже они нашли маленькую табличку, которая содержит предложение, написанное на том же самом языке с переводом на греческий язык. Используя табличку, они смогли прочитать первоначальную рукопись. Какую атаку использовали археологи?

3. Алиса может использовать только аддитивный шифр на ее компьютере, чтобы передать сообщение другу. Она думает, что сообщение будет более безопасно, если она зашифрует его два раза, каждый раз с различным ключом. Действительно ли она права? Обоснуйте ваш ответ.
4. Алиса хочет передать длинное сообщение. Она использует моноалфавитный шифр подстановки. Она думает, что если она сожмет сообщение, это может защитить текст от атаки Евы по частоте отдельных букв. Помогает ли сжатие? Должна ли она сжать сообщение, прежде чем зашифрует его или после этого? Обоснуйте ваш ответ.
5. Алиса часто должна зашифровывать исходный текст, использующий вместе буквы (от `a` до `z`) и цифры (от `0` до `9`).
    - Если она применяет аддитивный шифр, что является множеством ключей? Какие будут модули?
    - Если она применяет мультипликативный шифр, что является множеством ключей? Какие будут модули?
    - Если она применяет аффинный шифр, что является множеством ключей? Какие будут модули?
6. Предположим, что к исходному тексту добавляются пробелы, точки и знаки вопроса, чтобы увеличить множество ключей элементарных шифров.
    - Каково множество ключей, если используется аддитивный шифр?
    - Каково множество ключей, если используется мультипликативный шифр?
    - Каково множество ключей, если используется аффинный шифр?
7. Алиса и Боб решили игнорировать принципы Керкгоффса и скрывают тип шифра, который они используют.
    - Как может Ева понять, использовались ли шифр подстановки или шифр перестановки?
    - Если Ева знает, что использованный шифр — шифр подстановки, как может она определить, был ли он

аддитивным, мультипликативным или аффинным шифром?
    - Если Ева знает, что использованный шифр — шифр перестановки, как она может определить размер секции (`m`)?
8. В каждом из следующих шифров — какое максимальное число символов может быть изменено в зашифрованном тексте, если в исходном тексте изменен только единственный символ?
    - Аддитивный
    - Мультипликативный
    - Аффинный
    - Виженера
    - Автоключевой
    - Одноразовый блокнот
    - Роторный
    - "Энигма"
9. В каждом из следующих шифров — какое максимальное число символов будет изменено в зашифрованном тексте, если в исходном тексте изменен только один символ?
    - Одиночная перестановка
    - Двойная перестановка
    - Плейфеер
10. Для каждого из следующих шифров определите, является ли он шифром потока или блочным шифром. Обоснуйте ваши ответы.
    - Плейфеер
    - Автоключ
    - Одноразовый блокнот
    - Ротор
    - "Энигма"
11. Зашифруйте сообщение `"this is exercise"` (`"это — упражнение"`), используя один из следующих шифров. Игнорируйте пробелы между словами. Расшифруйте сообщение, чтобы получить первоначальный исходный текст.
    - Аддитивный шифр с ключом `= 20`
    - Мультипликативный шифр с ключом `= 15`
    - Аффинный шифр с ключом `= (15, 20)`
12. Зашифруйте сообщение `"the house is being sold tonight"` (`"дом продан сегодня вечером"`), используя один из следующих шифров. Игнорируйте пространство

между словами. Расшифруйте сообщение, чтобы получить исходный текст.

- Шифр Виженера с ключом: "dollars"
- Шифр с автоматическим ключом = 7
- Шифр Плейфера с ключом, созданным в тексте (см. рис. 4.13)

13. Используйте шифр Виженера с ключевым словом "*HEALTH*", чтобы зашифровать сообщение "*Life* is full surprises" ("Жизнь полна сюрпризов").
14. Используйте шифр Плейфера, чтобы зашифровать сообщение "The key hidden under the *door* pad" ("ключ спрятан под ковриком у двери"). Ключ засекречивания можно составить, заполняя первую и вторую часть строки со словом "*GUIDANCE*" и заполняя остальную часть матрицы с остальной частью алфавита.
15. Используйте шифр Хилла, чтобы зашифровать сообщение "We live in an *insecure* world" ("Мы живем в опасном мире"). Применить следующий ключ:

$$\mathbf{K} = \begin{pmatrix} 03 & 02 \\ 05 & 07 \end{pmatrix}$$

16. Джон читает тайную книгу введения в криптографию. В одной части книги автор дает зашифрованный текст "CIW" и двумя параграфами позже говорит читателю, что это — ключ сдвига, и исходный текст — "YES" ("да"). В следующей лекции герой нашел табличку с выгравированным на ней текстом "XV1EWYW1". Джон немедленно разгадал фактическое значение зашифрованного текста. Какой тип атаки предпринял Джон? Каков исходный текст?
17. Ева тайно получает доступ к компьютеру Алисы и, используя ее шифр, печатает "abcdefghij"; на экране появилось "CABDEHEGIJ". Предположим, Ева знает, что Алиса использует ключевой шифр перестановки. Ответьте на следующие вопросы:
    - Какую атаку предпринимает Ева?
    - Каков размер ключа перестановки?
18. Используйте атаку грубой силы, чтобы расшифровать следующее сообщение, зашифрованное Алисой, применяя аддитивный шифр. Предположим, что Алиса всегда использует ключ, связанный с ее

днем рождения, который приходится на 13 -е число месяца.

NCJAEZRCLASJLYODEPRLYZRCLASJLCPEHZDTOPD

19. Используйте атаку грубой силы, чтобы расшифровать следующее сообщение. Предположите, что Вы знаете: шифр — аффинный и исходный текст "ab" зашифрован "GL".

XPALASXYFGFUKPXUSOGEUTKCDGFXANMGNVS

20. Используйте атаку частоты отдельных букв, чтобы расшифровать следующее сообщение. Предположите, что Вы знаете, что оно зашифровано с применением моноалфавитного шифра подстановки.

ONHOVEJHWOBEVGWOCBWHNUGBLHGBGR

21. Предположим, что знаки препинания (точки, вопросительные знаки и пробелы) складываются с алфавитом шифрования шифра Хилла, потом для шифрования и дешифрования используются ключевые матрицы 2×2 в Z_{29}.
    - Найдите общее количество возможных матриц.
    - Доказано, что общее количество обратимых матриц — $(N^2 - 1)(N^2 - N)$, где N — число размера алфавита. Найдите множество ключей шифра Хилла, используя этот алфавит.
22. Используйте атаку частоты отдельных букв, чтобы взломать следующий зашифрованный текст. Предположите, что вы знаете, что он был создан с использованием аддитивного шифра.

OTWEWNGWCBPQABIZVQAPMLJGZWTTQVOBQUMAPMIDG
EQVBMZLZIXMLAXZQVOQVLMMXAVWEIVLLIZSNZWAB
JQZLWNLMTQOPBVIUMLGWCBPAEQNBTGTMNBBPMVMAB

23. Используйте тест Казиского и атаку частоты отдельных букв, чтобы нарушить следующий зашифрованный текст. Предположите, что вы знаете, что он был создан шифром Виженера.

OTWEWNGWCBPQABIZVQAPMLJGZWTTQVOBQUMAPMIDGZ
EQVBMZLZIXMLAXZQVOQVLMMXAVWEIVLLIZSNZWAB
JQZLWNLMTQOPBVIUMLGWCBPAEQNBTGTMNBBPMVMAB

24. Ключ шифрования в шифре перестановки — `(3, 2, 6, 1, 5, 4)`. Найдите ключ дешифрования.
25. Покажите матричное представление ключа шифрования перестановки с ключом `(3, 2, 6, 1, 5, 4)`. Найдите матричное представление ключа дешифрования.
26. Даны исходный текст `"letusmeetnow"` и соответствующий зашифрованный текст `"HBCDFNOPIKLB"`. Известно, что алгоритм — шифр Хилла, но вы не знаете размер ключа. Найдите ключевую матрицу.
27. Шифры Хилла и мультипликативные шифры очень похожи. Шифры Хилла — блочные шифры, использующие умножение матриц; мультипликативные шифры — шифры потока, использующие скалярное умножение.
    - Определите блочный шифр, который, подобно аддитивному шифру, использует сложение матриц.
    - Определите блочный шифр, который, подобно аффинному шифру, использует умножение и сложение матриц.
28. Определите новый шифр потока. Шифр является аффинным, но ключи зависят от позиции символа в исходном тексте. Если символ исходного текста будет зашифрован в позиции `i`, мы можем найти ключи:
    - Мультипликативный ключ — `(i mod 12)` элемент в $Z_{26}*$.
    - Аддитивный ключ — `(i mod 26)` элемент в Z_{26}.

    Зашифруйте сообщение `"cryptography is fun"` (`"криптография — забавно"`), используя этот новый шифр.

29. Предположим, что для шифра Хилла исходный текст является мультипликативной единичной матрицей (`I`). Найдите отношения между ключом и зашифрованным текстом. Используйте результат вашего исследования и попытайтесь атаковать выборку исходного текста, использующего шифр Хилла.
30. Atbash был популярным шифром среди Библейских авторов (VI

век до нашей эры). В Atbash "A" — шифровалось буквой "Z", "B" был зашифрован буквой "Y", и так далее. Аналогично "Z" был зашифрован как "A", "Y" зашифрован как "B", и так далее. Предположим, что алфавит разделен на две половины и буквы в первой половине зашифрованы как буквы во второй и наоборот. Найдите тип шифра и ключа. Зашифруйте сообщение "упражнение", используя Atbash-шифр.

31. В шифре Полибиуса (Polybius — римский историк, живший в IV веке до нашей эры) каждая буква зашифрована как два целых числа. Ключ — матрица символов 5×5, как в шифре Плейфера. Исходный текст — матрица символов, зашифрованный текст — эти два целых числа (каждое между 1 и 5), представляющие столбцы и строки. Зашифруйте сообщение "An *exercise*" ("упражнение"), используя шифр Полибиуса (Polybius) со следующим ключом:

	1	2	3	4	5
1	z	q	p	f	e
2	y	r	o	g	d
3	x	s	n	h	c
4	w	t	m	i/j	b
5	v	u	l	k	a

Алгебраические структуры

Мы постараемся здесь подготовить читателя к следующим нескольким лекциям, в которых рассматриваются современные шифры с симметричным ключом, основанные на алгебраических структурах. Эта лекция имеет несколько целей: рассмотреть понятие алгебраических структур; определить и привести некоторые примеры алгебраических групп; определить и привести некоторые примеры алгебраических колец.

Следующие лекции посвящены обсуждению современных симметрично-ключевых *блочных шифров*, которые выполняют операции с `n` -битовыми словами. Понимание и анализ этих шифров требуют некоторого знания разделов современной алгебры, называемых алгебраическими структурами. В начале этой лекции делается обзор алгебраических структур, а затем показывается, как выполнить сложение или умножение с `n` -битовыми словами.

Алгебраические структуры

В лекциях 2-3 мы обсуждали некоторые множества чисел, таких как Z, Z_n, Z_{n^*}, $Z_{p^\wedge}$ и Z_{p^*}. Криптография требует, чтобы были заданы множества целых чисел, и операции, определенные для них. Комбинация множеств и операций, которые могут быть применены к элементам множества, называются алгебраической структурой. В этой лекции мы определим три общих алгебраических структуры: группы, кольца и поля (рис. 5.1).

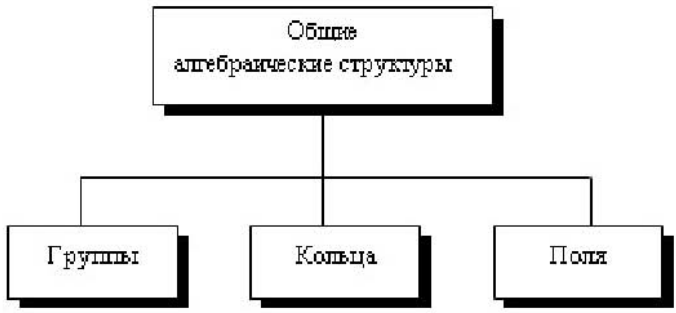


Рис. 5.1. Общие алгебраические структуры

Группы

Группа (G) — набор элементов с *бинарной операцией* "•" обладает четырьмя свойствами (или удовлетворяет *аксиомам*), которые будут перечислены ниже.

Коммутативная группа, также называемая абелева, — группа, в которой оператор обладает теми же четырьмя свойствами для групп плюс дополнительным — коммутативностью. Эти пять свойств определены ниже

- Замкнутость. Если a и b — элементы G, то c = a • b — также элемент G. Это означает, что результат применения операции с любыми двумя элементами множества есть элемент этого множества.
- Ассоциативность. Если a, b и c — элементы G, то верно (a• b) • c = a• (b •c) Другими словами, не имеет значения, в каком порядке мы применяем операцию более чем с двумя элементами.
- Коммутативность. Для всех a и b в G мы имеем a • b = b • a. Обратите внимание, что это свойство должно быть верно только для коммутативной группы.
- Существование нейтрального элемента. Для всех элементов в G существует элемент e, который называется нейтральным элементом, такой, что e • a = a • e = a.
- Существование инверсии. Для каждого a в G существует элемент a', называемый инверсией, такой, что a • a' = a' • a = e.

Рисунок 5.2 иллюстрирует понятие группы.

Приложение

Хотя группа включает единственный оператор, свойства, присущие каждой операции, позволяют использование пары операций, если они — инверсии друг друга. Например, если определенный выше оператор

— сложение, то группа поддерживает и сложение, и вычитание, ибо вычитание и сложение — аддитивно инверсные операции. Это также верно для умножения и деления. Однако группа может поддержать только сложение/вычитание или умножение/деление, но не оба сочетания операторов одновременно.

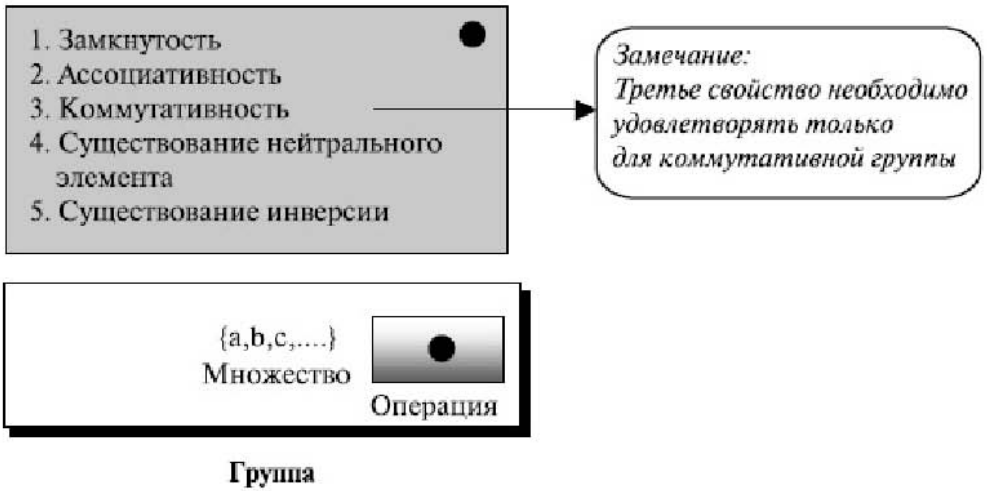


Рис. 5.2. Группа

Пример 5.1

Множество целых чисел, входящих в вычет с оператором сложения, $G = <Z_n, +>$, является коммутативной группой. Мы можем выполнить сложение и вычитание на элементах этого множества, не выходя за его пределы.

Проверим эти свойства.

1. Замкнутость удовлетворяется. Результат сложения двух целых чисел в Z_n — другое целое число в Z_n.
2. Ассоциативность удовлетворяется. Результат $4 + (3 + 2)$ тот же самый, что в случае $(4 + 3) + 2$.
3. *Коммутативность* удовлетворяется. Мы имеем $3 + 5 = 5 + 3$.
4. Нейтральный элемент — 0. Мы имеем $3 + 0 = 0 + 3 = 3$.
5. Каждый элемент имеет аддитивную *инверсию*. *Инверсия* элемента

— его дополнение. Например, *инверсия* 3 — это −3 (n − 3 в Z_n), и *инверсия* −3 — это 3. *Инверсия* позволяет нам выполнять вычитание на множестве.

Пример 5.2

Множество Z_{n*} с оператором умножения G = <Z_{n*}, > , является также абелевой группой. Мы можем выполнить умножение и деление на элементах этого множества, не выходя за его пределы. Это облегчает проверку первых трех свойств. Нейтральный элемент равен 1. Каждый элемент имеет *инверсию*, которая может быть найдена согласно расширенному *алгоритму Евклида*.

Пример 5.3

Хотя мы обычно представляем группу как множество чисел с обычными операторами, такими, как сложение или вычитание, определения группы позволяют нам определять любое множество объектов и операций, которые удовлетворяют вышеупомянутым свойствам. Определим множество G = <{a, b, c, d,}, •> и операцию, показанную с помощью таблицы 5.1.

Таблица 5.1. *Таблица операции* для примера 5.3

	a	b	c	d
a	a	b	c	d
b	b	c	d	a
c	c	d	a	b
d	d	a	b	c

Это — абелева группа. Все пять свойств удовлетворены.

1. Замкнутость удовлетворена. Применение оператора на любой паре элементов дает в результате другой элемент этого множества.
2. Ассоциативность также удовлетворена. Чтобы доказать это, мы должны проверить свойство для любой комбинации из трех элементов. Например, `(a+ b) + c = a+ (b + c) = d`.
3. Операция *коммутативна*. Мы имеем `a + b = b + a`.
4. Группа имеет нейтральный элемент, которым является `a`.
5. Каждый элемент имеет *инверсию*. Обратные пары могут быть найдены. В таблице они указаны теневыми элементами в каждой строке. Пары — `(a, a), (b, d), (c, c)`.

Пример 5.4

В группе элементы в множестве не обязательно должны быть числами или объектами; они могут быть правилами, отображениями, функциями или действиями. Очень интересная группа — группа подстановок. Множество всех перестановок и оператор является композицией: применения одной перестановки за другой. Рисунок 5.3 показывает композиции двух перестановок, которые перемещают три входных сигнала, чтобы создать три выходных сигнала.

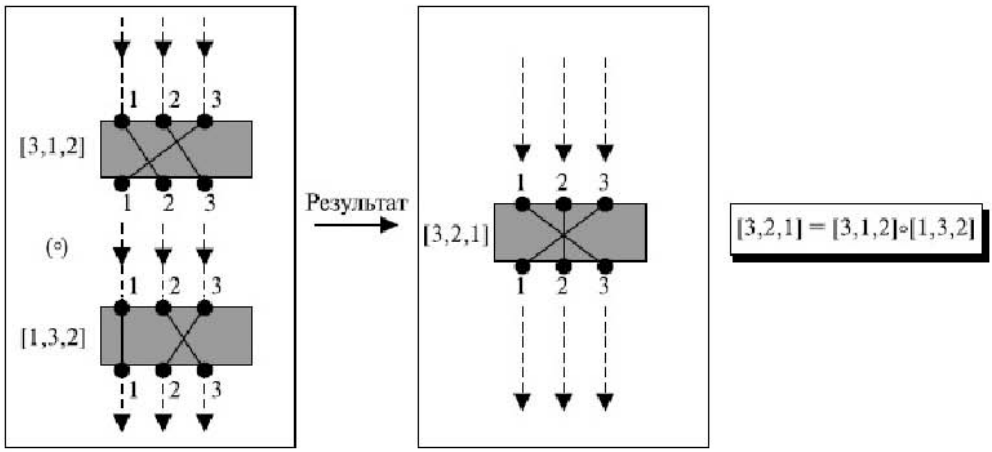


Рис. 5.3. Композиции перестановок (Пример 5.4)

Входные сигналы и выходные сигналы могут быть символами (лекции 2-3) или битами. Мы изобразили каждую перестановку прямоугольником, внутри которого показано, где исходящий входной

сигнал и индекс (`1,2,3`) определяет выходной сигнал. Композиция состоит из двух перестановок одна за другой. При трех входных сигналах и трех выходных сигналах может быть `3!` или `6` различных перестановок. Таблица 5.2 дает определение этого оператора. Первая строка — первая перестановка; первый столбец — вторая перестановка. Результат содержится на пересечении.

В этом случае удовлетворены только четыре свойства; поэтому группа — не абелева.

1. Замкнутость удовлетворена.
2. Ассоциативность также удовлетворена. Чтобы доказать это, мы должны проверить свойство для любой комбинации из трех элементов.
3. Свойство коммутативности не удовлетворено. Это может быть легко проверено, но мы оставим это для упражнения.
4. Множество имеет нейтральный элемент `[1 2 3]` (перестановка отсутствует). Эти элементы показаны другим цветом.
5. Каждый элемент имеет *инверсию*. Обратные пары могут быть найдены, если использовать нейтральные элементы.

Таблица 5.2. *Таблица операции* для группы перестановок

	[1 2 3]	[1 3 2]	[2 1 3]	[2 3 1]	[3 1 2]	[3 2 1]
[1 2 3]	[1 2 3]	[1 3 2]	[2 1 3]	[2 3 1]	[3 1 2]	[3 2 1]
[1 3 2]	[1 3 2]	[1 2 3]	[2 3 1]	[2 1 3]	[3 2 1]	[3 1 2]
[2 1 3]	[2 1 3]	[3 1 2]	[1 2 3]	[3 2 1]	[1 3 2]	[2 3 1]
[2 3 1]	[2 3 1]	[3 2 1]	[1 3 2]	[3 1 2]	[1 2 3]	[2 1 3]
[3 1 2]	[3 1 2]	[2 1 3]	[3 2 1]	[1 2 3]	[2 3 1]	[1 3 2]
[3 2 1]	[3 2 1]	[2 3 1]	[3 1 2]	[1 3 2]	[2 1 3]	[1 2 3]

Пример 5.5

В предыдущем примере мы показали, что множество перестановок с композицией операций — группа. Поэтому использование двух перестановок (одна за другой) не могут усилить безопасность шифра. Мы сможем всегда найти перестановку, которая может сделать ту же

самую операцию, используя свойства замкнутости.

Конечная группа

Группа называется конечной группой, если множество имеет конечное число элементов; иначе это — бесконечная группа.

Порядок группы

Порядок группы, G, — это число элементов в группе. Если группа не конечна, ее порядок бесконечен; если конечна, порядок конечен.

Подгруппы

Подмножество H группы G — подгруппа G, если само H — группа относительно операции на G. Другими словами, если G = <S, •> — группа, то H = <T, •> — группа для той же самой операции, и T — непустое подмножество S, то H — *подгруппа* G. Вышеупомянутое определение подразумевает, что:

1. если a и b — члены обеих групп, то c = a • b — также элемент обеих групп;
2. для группы и подгруппы имеется один и тот же нейтральный элемент;
3. если этот элемент принадлежит обеим группам, *инверсия* a — также элемент обеих групп;
4. группа, полученная с помощью нейтрального элемента G, H = <{e}, •>, является *подгруппой* G ;
5. каждая группа — *подгруппа* самой себя.

Пример 5.6

Является ли группа H = $<Z_{10}, +>$ *подгруппой* группы G = $<Z_{12}, +>$?

Решение

Ответ — нет. Хотя H — подмножество G, операции, определенные для этих двух групп, различны. Операция H — сложение по модулю 10 ; операция в G — сложение по модулю 12.

Циклические подгруппы

Если *подгруппа* группы может быть сгенерирована, используя возведение в степень элемента, то такая *подгруппа* называется циклической подгруппой. Термин возведение в степень здесь означает многократное применение к элементу групповой операции:

$$a^n \rightarrow a \bullet a \bullet \ldots \bullet a(\text{n раз})$$

Множество, полученное в результате этого процесса, обозначается в тексте как <a>. Обратите внимание также, что $a^0 = e$.

Пример 5.7

Из группы $G = < Z_6, +>$ могут быть получены четыре циклических подгруппы. Это $H_1 = <\{0\},+>$, $H_2 =<\{0, 2, 4\}, +>$, $H_3 = <\{0, 3\}, +>$ и $H_4 = G$. Заметим, что когда операция — сложение, то a^n означает умножение n на a. Заметим также, что во всех этих группах операция — это сложение по модулю 6. Ниже показано, как мы находим элементы этих циклических *подгрупп*.

а. Циклическая *подгруппа*, сгенерированная из 0, — это H_1, имеет только один элемент (нейтральный элемент).

$0^0 \bmod 6 = 0$ (остановка, далее процесс повторяется).

б. Циклическая *подгруппа*, сгенерированная на основе 1, — это H_4, которая есть сама группа G.

$1^0 \bmod 6 = 0$
$1^1 \bmod 6 = 1$
$1^2 \bmod 6 = (1 + 1) \bmod 6 = 2$
$1^3 \bmod 6 = (1 + 1 + 1) \bmod 6 = 3$

$1^4 \bmod 6 = (1 + 1 + 1 + 1) \bmod 6 = 4$

$1^5 \bmod 6 = (1 + 1 + 1 + 1 + 1) \bmod 6 = 5$(остановка, далее процесс повтор

в. Циклическая *подгруппа*, сгенерированная на основе 2, — это H_2, которая имеет три элемента: 0, 2, и 4.

$2^0 \bmod 6 = 0$

$2^1 \bmod 6 = 2$

$2^2 \bmod 6 = (2 + 2) \bmod 6 = 4$ (остановка, далее процесс повторяется)

г. Циклическая *подгруппа*, сгенерированная на основе 3, — это H_3, которая имеет два элемента: 0 и 3.

$3^0 \bmod 6 = 0$

$3^1 \bmod 6 = 3$ (остановка, далее процесс повторяется)

д. Циклическая *подгруппа*, сгенерированная на основе 4, — H_2 ; это — не новая *подгруппа*.

$4^0 \bmod 6 = 0$

$4^1 \bmod 6 = 4$

$4^2 \bmod 6 = (4 + 4) \bmod 6 = 2$ (остановка, далее процесс повторяется)

е. Циклическая *подгруппа*, сгенерированная на основе 5, — это H_4, она есть сама группа G.

$5^0 \bmod 6 = 0$

$5^1 \bmod 6 = 5$

$5^2 \bmod 6 = 4$

$5^3 \bmod 6 = 3$

$5^4 \bmod 6 = 2$

$5^5 \bmod 6 = 1$ (остановка, далее процесс повторяется)

Пример 5.8

Из группы $G = < Z_{10*}, \times >$ можно получить три циклических подгруппы. G имеет только четыре элемента: 1, 3, 7 и 9.

Циклические подгруппы — $H_1 = \{1\}, \times$, $H_2 = \{1, 9\}, \times$ и $H_3 = G$. Ниже показано, как мы находим элементы этих *подгрупп*.

а. Циклическая *подгруппа*, сгенерированная на основе 1, — это H_1. *Подгруппа* имеет только один элемент, а именно — нейтральный.

$1^0 \bmod 10 = 1$ (остановка, далее процесс повторяется)

б. Циклическая *подгруппа*, сгенерированная на основе 3, — это H_3, которая есть группа G.

$3^0 \bmod 10 = 1$
$3^1 \bmod 10 = 3$
$3^2 \bmod 10 = 9$
$3^3 \bmod 10 = 7$ (остановка, далее процесс повторяется)

в. Циклическая *подгруппа*, сгенерированная на основе 7, — это H_3, которая есть группа G.

$7^0 \bmod 10 = 1$
$7^1 \bmod 10 = 7$
$7^2 \bmod 10 = 9$
$7^3 \bmod 10 = 3$ (остановка, далее процесс повторяется)

г. Циклическая *подгруппа*, сгенерированная на основе 9, — это H_2. *Подгруппа* имеет только два элемента.

$9^0 \bmod 10 = 1$
$9^1 \bmod 10 = 9$ (остановка, далее процесс повторяется)

Циклические группы

Циклическая группа — группа, которая является собственной циклической *подгруппой*. В примере 5.7 группа G имеет циклическую подгруппу $H_5 = G$. Это означает, что группа G — циклическая группа. В этом случае элемент, который генерирует циклическую подгруппу,

может также генерировать саму группу. Этот элемент далее именуется "генератор". Если g — генератор, элементы в конечной циклической группе могут быть записаны как

$\{e, g, g^2, \ldots, g^{n-1}\}$, где $g^n = e$.

Заметим, что циклическая группа может иметь много генераторов.

Пример 5.9

а. Группа $G = \langle Z_6, + \rangle$ — циклическая группа с двумя генераторами, $g = 1$ и $g = 5$.

б. Группа $G = \langle Z_{10*}, \times \rangle$ — циклическая группа с двумя генераторами, $g = 3$ и $g = 7$.

Теорема Лагранжа

Теорема Лагранжа показывает отношение между порядком группы к порядку ее подгруппы. Предположим, что G — группа и H — *подгруппа* G. Если порядок G и H — |G| и |H|, соответственно, то согласно этой теореме |H| делит |G|. В примере 5.7 $|G| = 6$. Порядок подгруппы — $|H1| = 1$, $|H2| = 3$, $|H3| = 2$ и $|H4| = 6$. Очевидно, все эти порядки есть делители 6.

Теорема Лагранжа имеет очень интересное приложение. Когда дана группа G и ее порядок |G|, могут быть легко определены порядки потенциальных *подгрупп*, если могут быть найдены делители. Например, порядок группы $G = \langle Z_{17}, + \rangle$ — это |17|. Делители 17 есть 1 и 17. Это означает, что эта группа может иметь только две подгруппы — нейтральный элемент и $H_2 = G$.

Порядок элемента

Порядок элемента в группе ord (a) (порядок (а)) является наименьшим целым числом n, таким, что $a^n = e$. Иными словами:

порядок элемента — порядок группы, которую он генерирует.

Пример 5.10

a. В группе G = $<Z_6, +>$, порядки элементов: порядок ord(0) = 1, порядок ord (1) = 6, порядок ord (2) = 3, порядок ord (3) = 2, порядок ord (4) = 3, порядок ord (5) = 6.

b. В группе G = $<Z_{10}, * >$, порядки элементов: порядок ord (1) = 1, порядок ord (3) = 4, порядок ord (7) =4, порядок (9) = 2.

Кольцо

Кольцо, обозначенное как $R = \{...\}, \bullet, \perp$, является алгебраической структурой с двумя операциями. Первая операция должна удовлетворять всем пяти свойствам, требуемым для абелевой группы. Вторая операция должна удовлетворять только первым двум свойствам абелевой группы. Кроме того, вторая операция должна быть распределена с помощью первой. Дистрибутивность означает, что для всех a, b и c элементов из R мы имеем $a \perp (b \bullet c) = (a \perp b) \bullet (a \perp c)$ и $(a \bullet b) \perp c = (a \perp c) \bullet (b \perp c)$. *Коммутативное* кольцо — кольцо, в котором *коммутативное* свойство удовлетворено и для второй операции. Рисунок 5.4 показывает кольцо и *коммутативное* кольцо.

Дополнительное замечание

Кольцо включает две операции. Однако вторая операция может не соответствовать третьему и четвертому свойствам. Другими словами, первая операция — фактически операция пары операций, таких как сложение и вычитание; вторая операция может содержать единственную операцию, например умножение, но может не содержать деление.

Пример 5.11

Множество $\mathbb{Z}$ с двумя операциями — сложением и умножением — является коммутативным кольцом, которое обозначается $R = Z, +, \times$. Сложение удовлетворяет всем пяти свойствам; умножение удовлетворяет только трем свойствам.

Рис. 5.4. Кольцо

Умножение дистрибутивно с помощью сложения. Например, $5 \times (3 + 2) = (5 \times 3) + (5 \times 2) = 25$. Хотя, мы можем выполнить на этом множестве сложение и вычитание и умножение, но не деление. Деление не может применяться в этой структуре, потому что оно приводит к элементу из другого множества. Результат деления 12 на 5 есть $2,4$, и он не находится в заданном множестве.

Поле

Поле, обозначенное $F = \{...\}, \bullet, \perp$ — *коммутативное* кольцо, в котором вторая операция удовлетворяет всем пяти свойствам, определенным для первой операции, за исключением того, что нейтральный элемент первой операции (иногда называемый нулевой элемент) не имеет инверсии. Рисунок 5.5 показывает поле.

Рис. 5.5. Поле

Дополнительное замечание

Поле — структура, которая поддерживает две пары операций, используемые в математике: сложение/вычитание и умножение/деление. Есть одно исключение: не разрешено деление на нуль.

Конечные поля

Хотя общее определение касается полей бесконечного порядка, в криптографии используются экстенсивно только конечные поля. Конечное поле — поле с конечным числом элементов — является очень важной структурой в криптографии. Галуа показал что поля, чтобы быть конечными, должны иметь число элементов p^n, где p — простое, а n — положительное целое число. Конечные поля обычно называют полями Галуа и обозначают как $GF(p^n)$.

Поле Галуа, $GF(p^n)$, — конечное поле с p^n элементами.

Поля GF (p)

Когда $n = 1$, мы имеем поле $GF(p)$. Это поле может быть множеством Z_p, $(0, 1, \ldots p-1)$ с двумя арифметическими

операциями (сложение и умножение). Любой элемент в этом множестве имеет аддитивную *инверсию*, и элементы, отличные от нуля, имеют мультипликативную *инверсию* (мультипликативная *инверсия* для 0 отсутствует).

Пример 5.12

Очень общее поле в этой категории — GF (2) с множеством {0,1} и двумя операциями, сложением и умножением, как показано на рисунке 5.6.

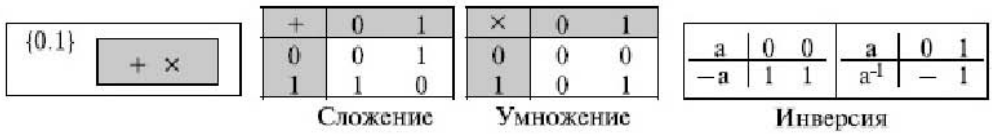


Рис. 5.6. Поле GF (2)

Есть несколько моментов, которые следует отметить в определении этого поля. Первый: множество имеет только два элемента, которые являются двоичными цифрами или битами (0 и 1). Второй: операция сложения — фактически ИСКЛЮЧАЮЩЕЕ ИЛИ (XOR), операция, которую мы используем с двумя двоичными цифрами. Третий: операция умножения — AND, операция, которую мы используем с двумя двоичными цифрами. Четвертый: сложение и операции вычитания — те же самые (операция XOR). Пятый: умножение и операции деления — те же самые (ОПЕРАЦИЯ AND).

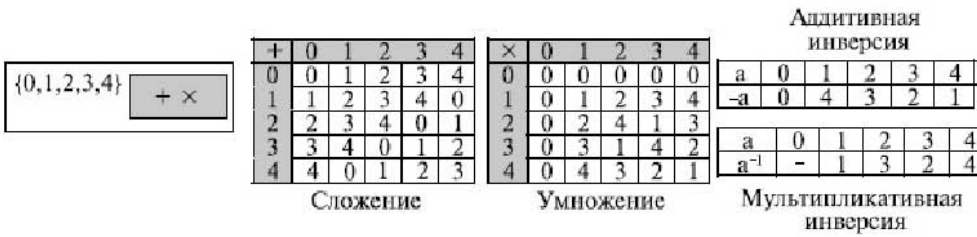
Сложение/вычитание в GF(2) — операция ИСКЛЮЧАЮЩЕЕ ИЛИ (XOR); умножение/деление — ОПЕРАЦИЯ И (AND).

Пример 5.13

Мы можем определить GF(5) на множестве Z_5 (5 простое) с операторами сложения и умножения, показанными на рис. 5.7.

Хотя мы можем использовать расширенный алгоритм Евклида, чтобы найти мультипликативные инверсии элементов в GF(5), проще составить таблицу умножения и находить каждую пару, произведение которой равняется 1. Это (1, 1), (2, 3), (3, 2) и (4, 4).

Заметим, что мы можем на этом множестве применить вычитание и умножение/деление (за исключением запрещенного деления на 0).



+	0	1	2	3	4
0	0	1	2	3	4
1	1	2	3	4	0
2	2	3	4	0	1
3	3	4	0	1	2
4	4	0	1	2	3

×	0	1	2	3	4
0	0	0	0	0	0
1	0	1	2	3	4
2	0	2	4	1	3
3	0	3	1	4	2
4	0	4	3	2	1

a	0	1	2	3	4
-a	0	4	3	2	1

a	0	1	2	3	4
a^{-1}	-	1	3	2	4

Рис. 5.7. Поле GF (5)

Поля GF(p в степени n)

В дополнение к полям GF(p) в криптографии мы также интересуемся полями GF(pn). Однако множества Z, Z_n, Z_{n*} и Z_p, которые мы использовали до сих пор с операциями сложения и умножения, не могут удовлетворить требованиям поля. Поэтому должны быть определены некоторые новые множества и некоторые новые операции на этих множествах. В следующей лекции мы рассматриваем очень полезное в криптографии поле GF(2n).

Итоги рассмотренных структур

Изучение трех алгебраических структур позволяет нам использовать множества, в которых могут применяться операции, подобные сложению/вычитанию и умножению/делению. Мы должны различать эти три структуры. Первая структура — группа, поддерживает одну пару связанных операций. Вторая структура — кольцо, поддерживает одну пару связанных операций и одну одиночную операцию. Третья структура — поле, поддерживает две пары операций. Таблица 5.3 может помочь нам увидеть эту разницу.

Таблица 5.3. Итоги определения алгебраических структур

Алгебраическая структура	Используемые операции	Используемые наборы целых чисел

Группа	(+ -) или (x /)	Z_n или Z_n^*
Кольцо	(+ -) и (x)	Z
Поле	(+ -) и (x /)	Z_p

Поля

Цели данной лекции: определить и привести некоторые примеры алгебраических полей; поговорить о таких операциях, как сложение, вычитание, умножение и деление с n-битовыми словами в современных блочных шифрах.

Поля GF(2n)

В криптографии мы часто должны использовать четыре операции (сложение, вычитание, умножение и деление). Другими словами, мы должны использовать поля. Однако когда мы работаем с компьютерами, положительные целые числа сохраняются в компьютере как n -битовые слова, в которых n является обычно 8, 16, 32, 64, и так далее. Это означает, что диапазон целых чисел — 0 до $2^n - 1$. Модуль — 2^n. Так что возможны два варианта, если мы хотим использовать поле.

1. Мы можем задействовать $GF(p)$ с множеством Z_p, где p — наибольшее простое число, меньшее, чем 2^n. Но эта схема неэффективна, потому что мы не можем использовать целые числа от p до $2^n - 1$. Например, если $n = 4$, то наибольшее простое число, меньшее, чем 2^4, — это 13. Это означает, что мы не можем использовать целые числа 13, 14 и 15. Если $n = 8$, наибольшее простое число, меньшее, чем 2^8, — это 251, так что мы не можем использовать 251, 252, 253, 254 и 255.

2. Мы можем работать в $GF(2^n)$ и использовать множество 2^n элементов. Элементы в этом множестве — n -битовые слова. Например, если $n = 3$, множество равно:

{000,001,010,011, 100,101,110,111}

Однако мы не можем интерпретировать каждый элемент как целое число от 0 до 7, потому что не могут быть применены обычные четыре операции (модуль 2^n — не простое число). Мы должны определить множество слов по 2 бита и две новых операции, которые удовлетворяют свойствам, определенным для поля.

Пример 6.1

Определим GF(2^2) поле, в котором множество имеет четыре слова по 2 бита: {00, 01, 10, 11}. Мы можем переопределить сложение и умножение для этого поля таким образом, чтобы все свойства этих операций были удовлетворены, как это показано на рис. 6.1.

⊕	00	01	10	11
00	00	01	10	11
01	01	00	11	10
10	10	11	00	01
11	11	10	01	00

Нейтральный элемент 00

⊗	00	01	10	11
00	00	00	00	00
01	00	01	10	11
10	00	10	11	01
11	00	11	01	10

Нейтральный элемент 01

Рис. 6.1. Пример поля GF(2 в степени 2)

Каждое слово — аддитивная *инверсия* себя. Каждое слово (кроме 00) имеет мультипликативную *инверсию*. Мультипликативные обратные пары — (01, 01) и (10, 11). Сложение и умножение определено в терминах полиномиалов.

Полиномы

Хотя мы можем непосредственно определить правила для операций сложения и умножения слов из двух бит, которые удовлетворяют свойства в GF(2^n), проще работать с полиномиальным степени n – 1 побитным представлением слов. Полиномиальное выражение степени n – 1 имеет форму

$$f(x) = a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \ldots\ldots + a_1x^1 + a_0x^0$$

где x^i назван термином " i -тый элемент", а a_i называется коэффициентом i - того элемента. Хотя мы знаем полиномы в алгебре, но при представлении n -битовых слов полиномами необходимо следовать некоторым правилам:

а. степень x определяет позицию бита в n -битовых слов. Это означает, что крайний левый бит находится в нулевой позиции (связан с x^0),

самый правый бит находится в позиции n–1 (связан с x^{n-1});

b. коэффициенты сомножителей определяют значение битов. Каждый бит принимает только значение 0 или 1, поэтому наши полиномиальные коэффициенты могут иметь значение 0 или 1.

Пример 6.2

Использование полиномов для предоставления слова из 8 бит (10011001) показано на рис. 6.2.

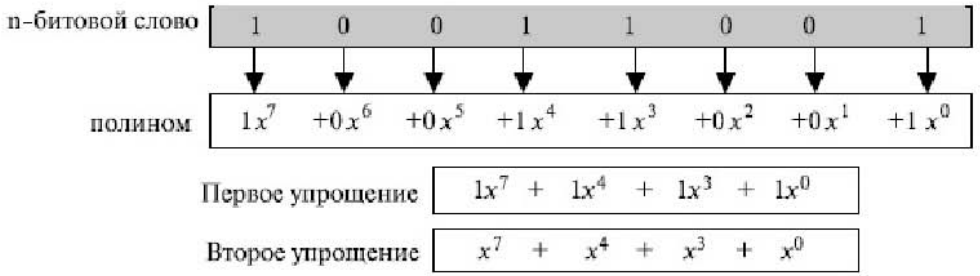


Рис. 6.2. Представление 8-ми битового слова полиномом

Заметим, что элемент полностью пропущен, если его коэффициент равен 0, и пропущен только коэффициент, если это 1. Также заметим, что элемент x^0 равен 1.

Пример 6.3

Чтобы найти слово на 8 битов, связанное с полиномом $X^5 + X^2 + X$, мы сначала восстановим пропущенные сомножители. Мы имеем n = 8, это означает *полином* степени 7. Расширенный *полином* имеет вид

$$0X^7 + 0X^6 + 1X^5 + 0X^4 + 0X^3 + 1X^2 + 1X^1 + 0X^0$$

Он связан со словом на 8 битов 00100110.

Операции

Обратите внимание, что любая операция на полиномах фактически

включает две операции: операции И коэффициентов двух полиномов. Другими словами, мы должны определить два поля: одно для коэффициентов и одно для полиномов. Коэффициенты равны `0` или `1` ; для этой цели мы можем использовать `GF(2)` -поле. Мы уже говорили о таком поле (см. пример 6.1). Для полиномов нам нужно поле $GF(2^n)$, которое мы коротко обсудим ниже.

Полиномы, представляющие n-битовые слова, используют два поля: GF(2) и $GF(2^n)$.

Модуль

Перед определением операций на полиномах мы должны поговорить о полиномах-модулях. Сложение двух полиномов никогда не создает *полином*, выходящий из множества. Однако умножение двух полиномов может создать *полином* со степенью большей, чем `n - 1`. Это означает, что мы должны делить результат на модуль и сохранять только остаток, как мы сделали в модульной арифметике. Для множеств полиномов в $GF(2^n)$ группа полиномов степени `n` определена как модуль. Модуль в этом случае действует как полиномиальное простое число. Это означает, что никакие полиномы множества не могут делить этот *полином*. Простое полиномиальное число не может быть разложено в полиномы со степенью меньшей, чем `n`. Такие полиномы называются неприводимые полиномы. Таблица 6.1 показывает примеры полиномов `1-5` степеней.

Для каждого значения степени часто есть более чем один неразлагаемый *полином*, — это означает, что когда мы определяем наш $GF(2^n)$, мы должны объявить, какой неприводимый *полином* мы используем как модуль.

Таблица 6.1. Список неприводимых полиномов

Степень	Неприводимый *полином*
1	$(x+1)x$
2	(x^2+x+1)
3	$(x^3+x^2+1)(x^3+x+1)$

4	$(x^4+x^3+x^2+x+1)(x^4+x^3+1)(x^4+x+1)$
5	$(x^5+x^2+1)(x^5+x^3+x^2+x+1)(x^5+x^4+x^3+x+1)(x^5+x^4+x^3+x^2+1)$ $(x^5+x^4+x^2+x+1)$

Сложение

Теперь определим операцию сложения для полиномов с коэффициентом в `GF(2)`. Операция сложения очень простая: мы складываем коэффициенты соответствующих элементов полинома в поле `GF(2)`. Обратите внимание, что сложение двух полиномов степени `n - 1` всегда дает *полином* со степенью `n - 1` — это означает, что мы не должны использовать вычитание из модуля их результата.

Пример 6.4

Произведем сложение $(x^5 + x^2 + x) \oplus (x^3 + x^2 + 1)$ в $GF(2^8)$. Мы используем символ $\oplus$ для обозначения полиномиального сложения. Ниже показана процедура

$$0x^7 + 0x^6 + 1x^5 + 0x^4 + 0x^3 + 1x^2 + 1x^1 + 0x^0 \oplus$$
$$0x^7 + 0x^6 + 0x^5 + 0x^4 + 1x^3 + 1x^2 + 0x^1 + 1x^0$$

$$0x^7 + 0x^6 + 1x^5 + 0x^4 + 1x^3 + 0x^2 + 1x^1 + 1x^0 \rightarrow x^5 + x^3 + x + 1$$

В упрощенном полиноме (показан справа) сохранены элементы с коэффициентом `1` и удалены элементы с коэффициентом `0`. Кроме того, удалены совпадающие элементы обоих полиномов, а несовпадающие сохраняются. Другими словами, x^5, x^3, и x^1 сохраняются, а x^2, который является совпадающим в этих двух полиномах, удален.

Пример 6.5

Поскольку сложение в `GF(2)` означает операцию *ИСКЛЮЧАЮШЕЕ ИЛИ* (*XOR*), мы можем получить результат *ИСКЛЮЧАЮШЕГО ИЛИ*

для этих двух слов бит за битом. В предыдущем примере $x^5 + x^2 + x$ есть 00100110, или *полином*, и $x^3 + x^2 + 1$ есть 00001101. Результат — 00101011 или, в полиномиальном обозначении, $x^5 + x^3 + x + 1$.

Аддитивный нейтральный элемент — тождество. Аддитивный нейтральный элемент полинома — нулевой *полином* (*полином* со всеми коэффициентами, равными нулю), потому что, прибавляя этот *полином* к самому себе, в результате получаем нулевой *полином*.

Аддитивная инверсия полинома с коэффициентами в GF(2) — сам *полином*. Это означает, что операция вычитания та же самая, что и операция сложения.

Сложение и операции вычитания на полиномах — та же самая операция.

Умножение

Умножение в полиномах — сумма умножения каждого элемента одного полинома с каждым элементом второго полинома. Однако необходимо отметить три особенности.

Первая: умножение коэффициента проводится в поле GF(2).

Вторая: умножение x^i на x^j дает результат x^{i+j}.

Третья: умножение может создать элементы со степенью большей, чем n–1, и это означает, что результат должен быть уменьшен с использованием полинома-модуля.

Сначала покажем, как умножить два полинома согласно вышеупомянутому определению. Позже будет дан более эффективный алгоритм, который может использоваться компьютерной программой.

Пример 6.6

Найдите результат $(x^5 + x^2 + x) \otimes (x^7 + x^4 + x^3 + x^2 + x)$ в

$GF(2^8)$ с неразлагаемым полиномом $(x^8 + x^4 + x^3 + x + 1)$. Обратите внимание, что для обозначения умножения двух полиномов используется символ $\otimes$.

Решение

Сначала умножаем эти два полинома так, как мы это делали в обычной алгебре. Обратите внимание, что в этом процессе пара элементов с равной степенью удаляется. Например, результат $x^9 + x^9$ полностью удален, потому что он нулевой *полином*, по причине, которую мы обсуждали раньше при рассмотрении операции сложения.

$$P_1 \otimes P_2 = x^5(x^7+x^4+x^3+x^2+x)+x^2(x^7+x^4+x^3+x^2+x)+x(x^7+x^4+x^3+x^2+x)$$
$$P_1 \otimes P_2 = x^{12}+x^9+x^8+x^7+x^6+x^9+x^6+x^5+x^4+x^3+x^8+x^5+x^4+x^3+x^2$$
$$P_1 \otimes P_2 = (x^{12}+x^7+x^2) \mod (x^8+x^4+x^3+x+1) = x^5+x^3+x^2+x+1$$

Чтобы найти конечный результат, разделим *полином* степени 12 на *полином* степени 8 (модуль) и сохраним только остаток. Процесс деления тот же самый, что и в обычной алгебре, но мы должны помнить, что здесь вычитание то же самое, что и сложение. Рисунок 6.3 показывает процесс деления.

$x^{12}+x^7+x^2$	$x^8+x^4+x^3+x+1$
$x^{12}+x^8+x^7+x^5+x^4$	x^4+1
$x^8+x^5+x^4+x^2$	
$x^8+x^4+x^3+x+1$	
$x^5+x^3+x^2+x+1$	Остаток

Рис. 6.3. Полиномиальное деление с коэффициентами в поле GF (2).

Мультипликативное тождество — всегда равно 1. Например, в $GF(2^8)$ мультипликативная *инверсия* — в побитном изображении 00000001.

Мультипликативная инверсия. Поиск мультипликативной *инверсии* требует привлечения расширенного *алгоритма Евклида*. *Алгоритм Евклида* должен быть применен к модулю и полиному, выполнение алгоритма является таким же, как и для целых чисел.

Пример 6.7

В $GF(2^4)$ найдите *инверсию* $(x^2 + 1) \bmod (x^4 + x + 1)$.

Решение

Мы используем расширенный евклидов алгоритм, как это показано в таблице 6.2:

Таблица 6.2. *Алгоритм Евклида* для упражнения 6.7

q	r_j	r_2	r	t_j	t_2	t
x^2+1	(x^4+x+1)	(x^2+1)	(x)	(0)	(1)	(x^2+1)
(x)	(x^2+1)	(x)	(1)	(1)	(x^2+1)	(x^3+x+1)
(x)	(x)	(1)	(0)	(x^2+1)	(x^3+x+1)	(0)
	(1)	(0)		(x^3+x+1)	(0)	

Это означает, что $(x^2 + 1)^{-1} \bmod (x^4 + x + 1)$ есть $(x^3 + x + 1)$. Ответ может быть проверен просто: надо перемножить эти два полинома и найти остаток. В этом случае результат деления на модуль равен

$$[(x^2 + 1) \otimes (x^3 + x + 1)] \mod (x^4 + x + 1) = 1$$

Пример 6.8

В $GF(2^8)$ найдите *инверсию* $(x^5) \bmod (x^8 + x^4 + x^3 + x + 1)$.

Решение

Будем использовать расширенный евклидов алгоритм, как это показано в Таблице 6.3:

Таблица 6.3. Евклидов алгоритм для примера 6.8

q	r_j	r_2	r	t_j	t_2
(x^3)	$(x^8+x^4+x^3+x+1)$	(x^5)	$(x^4+x^3+x^2+x+1)$	(0)	(1)

(x+1)	(x^5)	(x^4+x^3+x+1)	(x^3+x^2+1)	(1)	(x^3)
(x)	(x^4+x^3+x+1)	(x^3+x^2+1)	(1)	(x^3)	(x^4+x^3+1)
	(x^3+x^2+1)	(1)	(0)	(x^4+x^3+1)	($x^5+x^4+x^2+x$
	(1)	(0)		($x^5+x^4+x^2+x$)	(0)

Это означает, что $(x^5)^{-1} \bmod (x^8 + x^4 + x^3 + x + 1)$ есть $(x^5 + x^4 + x^2 + x)$.

Результат может быть легко проверен умножением этих двух полиномов и определением остатка деления по модулю.

$$[(x^5) \otimes (x^5 + x^4 + x^3 + x)] \mod (x^8 + x^4 + x^3 + x + 1) =$$

Умножение, использующее компьютер

Операция деления порождает проблему написания эффективной программы умножения двух полиномов. Лучший алгоритм для компьютерной реализации использует неоднократное умножение уменьшенного полинома на x. Например, вместо того чтобы находить результат $(x^2 \otimes P_2)$, программа находит результат $(x \otimes (x \otimes P_2))$. Преимущества этой стратегии будет обсуждаться далее, но сначала рассмотрим пример, чтобы проиллюстрировать алгоритм.

Пример 6.9

Найдите результат умножения $P1 = (x^5 + x^2 + x)$ на $P2 = (x^7 + x^4 + x^3 + x^2 + x)$ в поле $GF(2^8)$ с неприводимым полиномом $(x^8 + x^4 + x^3 + x + 1)$, используя алгоритм, изложенный выше.

Решение

Процесс показан в таблице 6.4. Мы сначала находим промежуточный результат умножения x^0, x^1, x^2, x^3, x^4 и x^5. Заметим, что необходимы только три составляющие произведения $x^m \otimes P_2$ для m от

0 до 5, каждое вычисление зависит от предыдущего результата.

Таблица 6.4. Эффективный алгоритм умножения, использующий полиномы (пример 6.9)

Степень	Операция	Новый результат	Вычитание
$x^0 \otimes P_2$		$x^7+x^4+x^3+x^2+x$	НЕТ
$x^1 \otimes P_2$	$x \otimes (x^7+x^4+x^3+x^2+x)$	x^5+x^2+x+1	ДА
$x^2 \otimes P_2$	$x \otimes (x^5+x^2+x+1)$	$x^6+x^3+x^2+x$	НЕТ
$x^3 \otimes P_2$	$x \otimes (x^6+x^3+x^2+x)$	$x^7+x^4+x^3+x^2$	НЕТ
$x^4 \otimes P_2$	$x \otimes (x^7+x^4+x^3+x^2)$	x^5+x+1	ДА
$x^5 \otimes P_2$	$x \otimes (x^5+x+1)$	x^6+x^2+x	НЕТ
$P_1 \times P_2 = (x^6+x^2+x)+(x^6+x^3+x^2+x)+(x^5+x^2+x+1) = x^5+x^3+x^2+x+1$			

Рассмотренный выше алгоритм имеет два преимущества. Первое — умножение полинома на x может быть выполнено простым сдвигом одного бита в n -битовом слове; операция может быть реализована на любом языке программирования. Второе — результат может быть использован, если максимальная степень полинома n–1. В этом случае сокращение может быть сделано просто с помощью применения операции *ИСКЛЮЧАЮШЕЕ ИЛИ* с заданным модулем. В нашем примере самая высокая степень — только 8. Мы можем разработать простой алгоритм для нахождения промежуточных результатов.

1. Если старший разряд предыдущего результата равен 0, тогда надо сдвинуть предыдущий результат на один бит влево.
2. Если старший бит предыдущего результата равен 1: а. надо сдвинуть на один бит влево, и б. применить к нему операцию *ИСКЛЮЧАЮШЕЕ ИЛИ* с модулем, исключив из этой операции старший разряд.

Повторим пример 6.9 для двоичной последовательности размером 8 бит. Пусть P_1 = 00100110, P_2 = 10011110, модуль = 100011010 (девять битов). Обозначим операцию *ИСКЛЮЧАЮШЕЕ ИЛИ* как $\oplus$. Пример приведен в таблице 6.5.

Таблица 6.5. Эффективное умножение с применением n-битового слова

Степень	*Операция сдвига* влево	*ИСКЛЮЧАЮШЕЕ ИЛИ*
$x^0 \otimes P_2$		10011110
$x^1 \otimes P_2$	00111100	(00111100) $\oplus$ (00011010) = 00100111
$x^2 \otimes P_2$	01001110	01001110
$x^3 \otimes P_2$	10011100	10011100
$x^4 \otimes P_2$	00111000	(00111000) $\oplus$ (00011010) = 00100011
$x^5 \otimes P_2$	01000110	01000110
$P_1 \otimes P_2$ = (000100110) $\oplus$ (01001110) $\oplus$ (01000110) = 00101111		

В этом случае для умножения этих двух полиномов нам надо только пять операций левого сдвига и четыре *ИСКЛЮЧАЮШЕЕ ИЛИ*. Вообще, для умножения двух полиномов степени $n-1$ необходимо максимально $(n-1)$ операций левого сдвига и $2n$ операций *ИСКЛЮЧАЮШЕЕ ИЛИ*.

Умножение полиномов в $GF(2^n)$ может быть выполнено с помощью операций левого сдвига и ИСКЛЮЧАЮЩЕЕ ИЛИ.

Пример 6.10

Поле $GF(2^3)$ состоит из 8 элементов. Покажем умножение и сложение таблиц для этого поля, используя неприводимый *полином* x^3+x^2+1. Мы будем оперировать с трехбитовым словом и полиномом. Заметим, что имеется два полинома третьей степени (см. таблицу 6.1). Другой *полином* $(x^3 + x + 1)$ для умножения имеет таблицу, полностью отличающуюся от первой. Таблица 6.6 показывает сложение. Затемненные клетки (нулевые) дают обратные пары для сложения.

Таблица 6.7 показывает умножение. Затемненные клетки (единичные) дают обратные пары при умножении.

Использование генератора

Иногда проще определить элементы поля $GF(2^n)$, используя генератор. В этом поле с неприводимым полиномом $f(x)$ и элементом поля a нужно удовлетворить отношение $f(a) = 0$. В частности, если g — генератор поля, то $f(g)=0$. Тогда можно доказать, что элементы поля могут быть сгенерированы как

$\{0, g, g, g^2, \ldots. g^n\}$, где $N = 2^n - 2$

Таблица 6.6. Сложение в поле GF(2 в степени 3)

⊕	000 (0)	001 (1)	010 (x)	011 (x+1)	100 (x^2)	101 (x^2+1)	110 (x^2 + x)	(x
000 (0)	000 (0)	001 (1)	010 (x)	011 (x+1)	100 (x^2)	101 (x^2+1)	110 (x^2 + x)	11 (x
001 (1)	001 (1)	000 (0)	011 (x+1)	010 (x^2)	101 (x^2+1)	100 (x^2+x)	111 (x^2+x+1)	11 +
010 (x)	010 (x)	011 (x+1)	000 (0)	001 (1)	110 (x^2 + x)	111 (x^2+x+1)	100 (x^2+x)	1((x
011 (x+1)	011 (x+1)	010 (x)	001 (1)	000 (0)	111 (x^2+x+1)	110 (x^2 + x)	101 (x^2+1)	1(
100 (x^2)	100 (x^2)	101 (x^2+1)	110 (x^2 + x)	111 (x^2+x+1)	000 (0)	001 (1)	010 (x)	01 (x
101 (x^2+1)	101 (x^2+1)	100 (x^2)	111 (x^2+x+1)	110 (x^2 + x)	001 (1)	000 (0)	011 (x+1)	01
110 (x^2 + x)	110 (x^2 + x)	111 (x^2+x+1)	100 (x^2)	101 (x^2+1)	010 (x)	011 (x+1)	000 (0)	0(
111 (x^2+x+1)	111 (x^2+x+1)	110 (x^2 + x)	101 (x^2+1)	100 (x^2)	011 (x+1)	010 (x)	001 (1)	0(

Таблица 6.7. Умножение в поле GF(2 в степени 3)

⊕	000 (0)	001 (1)	010 (x)	011 (x+1)	100 (x^2)	101 (x^2+1)	110 (x^2 + x)	111 (x^2+x+
000 (0)	000 (0)	000 (0)	000 (0)	000 (0)	000 (0)	000 (0)	000 (0)	000 (0)
001 (1)	000	001 (1)	010 (x)	011		101	110 (x^2	111

001 (1)	(0)	001 (1)	010 (x)	(x+1)		(x^2+1)	+ x)	(x^2+x+
010 (x)	000 (0)	010 (x)	100 (x)	110 (x^2 + x)	101 (x^2+1)	111 (x^2+x+1)	001 (1)	011 (x+1)
011 (x+1)	000 (0)	011 (x+1)	110 (x^2 + x)	101 (x^2+1)	001 (1)	010 (x)	111 (x^2+x+1)	100 (x^2
100 (x^2)	000 (0)	100 (x^2)	101 (x^2+1)	001 (1)	111 (x^2+x+1)	011 (x+1)	010 (x)	110 (x^2 + x)
101 (x^2+1)	000 (0)	101 (x^2+1)	111 (x^2+x+1)	010 (x)	011 (x+1)	110 (x^2 + x)	100 (x^2)	001 (1)
110 (x^2 + x)	000 (0)	110 (x^2 + x)	001 (1)	111 (x^2+x+1)	010 (x)	100 (x^2)	011 (x+1)	101 (x^2+1)
111 (x^2+x+1)	000 (0)	111 (x^2+x+1)	011 (x+1)	100 (x^2)	110 (x^2 + x)	001 (1)	101 (x^2+1)	010 (x)

Пример 6.11

Для генерирования элементов поля $GF(2^4)$ используйте *полином* $f(x) = x^4 + x + 1$.

Решение

Элементы 0, g^0, g^1, g^2 и g^3 могут быть сгенерированы достаточно просто, потому что в 4 *-битовом поле* они представлены 0, x^0, x^1, x^2 и x^3 (не требуется деления на *полином*). Элементы от g^4 до g^{14}, которые представляют g^4 до g^{14} от x^4 до x^{14}, нужно разделить на неприводимый *полином*. Для такого деления можно использовать *полином* $f(g) = g^4 + g + 1 = 0$. Применив это отношение, мы имеем $g^4 = -g-1$. поскольку сложение полей и вычитание полей — та же самая операция, $g^4 = g + 1$. Мы используем это отношение, чтобы найти значение всех элементов в виде 4 -битовых слов:

$0 = 0 = 0 = 0$ -> 0=(0000)
$g^0 = g^1 = g^1$ -> g^1 = (0010)

$g^2 = g^2 = g^2 \rightarrow g^2 = (0100)$
$g^3 = g^3 = g^3 \rightarrow g^3 = (1000)$
$g^4 = g^4 = g + 1 \rightarrow g^4 = (0011)$
$g^5 = g(g + 1) = g^2 + g \rightarrow g^5 = (0110)$
$g^6 = g(g^2 + g) = g^3 + g^2 \rightarrow g^6 = (1100)$
$g^7 = g(g^3 + g) = g^3 + g + 1 \rightarrow g^7 = (1011)$
$g^8 = g(g^3 + g + 1) = g^2 + 1 \rightarrow g^8 = (0101)$
$g^9 = g(g^2 + 1) = g^3 + g \rightarrow g^9 = (1010)$
$g^{10} = g(g^3 + g) = g^2 + g + 1 \rightarrow g^{10} = (0111)$
$g^{11} = g(g^2 + g + 1) = g^3 + g^2 + g \rightarrow g^{11} = (1110)$
$g^{12} = g(g^3 + g^2 + g) = g^3 + g^2 + g + 1 \rightarrow g^{12} = (1111)$
$g^{13} = g(g^3 + g^2 + g + 1) = g^3 + g^2 + 1 \rightarrow g^{13} = (1101)$
$g^{14} = g(g^3 + g^2 + 1) = g^3 + 1 \rightarrow g^{14} = (1001)$

Главная идея состоит в том, что вычисление элементов поля от g^4 до g^{14} сводится к использованию соотношения $g^4 = g + 1$ и предыдущих вычислений. Например,

$$g^{12} = g(g^{11}) = g(g^3 + g^2 + g) = g^4 + g^3 + g^2 = g^3 + g^2 + g + 1$$

После сокращения можно просто преобразовать степени в n -битовое слово. Скажем, $g^3 + 1$ эквивалентно 1001, потому что присутствуют элементы со степенью 0 и 3. Заметим, что элементы с одинаковой степенью при таком процессе вычисления отменяют друг друга. Например, $g^2 + g^2 = 0$.

Инверсии

Нахождение *инверсий* при использовании приведенного выше метода представления достаточно просто.

Аддитивные инверсии

Аддитивная *инверсия* каждого элемента — элемент непосредственно, потому что сложение и вычитание в этом поле — одна и та же операция, $g^3 = g^3$.

Мультипликативные инверсии

Найти мультипликативную *инверсию* каждого элемента также очень просто. Например, может найти мультипликативную *инверсию* элемента g^3, как показано ниже:

$$\left(g^3\right)^{-1} = g^{-3} = g^{12} = g^3 + g^2 + g + 1 \rightarrow (1111)$$

Заметим, что в этом случае степень рассчитывается по модулю $2^n - 1$, $2^4 - 1 = 15$.

Поэтому $-3 \bmod 15 = 12 \bmod 15$.

Можно легко доказать, что g^3 и g^{12} есть инверсные (обратные числа), потому что $g^3\ g^{12} = g^{15} = g^0 = 1$.

Сложение и вычитание

Сложение и вычитание — это одинаковые операции. Промежуточные результаты могут быть упрощены, как проиллюстрировано в следующем примере.

Пример 6.12

Этот пример показывает результаты операций сложения и вычитания:

a.

$$\left(g^3 + g^{12} + g^7\right) = g^3 + \left(g^3 + g^2 + g + 1\right) + \left(g^3 + g + 1\right) = g^3 + g^2 \rightarrow (1100)$$

b. $g^3 - g^6 = g^3 + g^6 = g^3 + (g^3 + g^2) = g^2 \rightarrow (0100)$

Умножение и деление

Умножение есть сложение степени по модулю $2^n - 1$. Деление — это умножение, которое использует мультипликативную *инверсию*.

Пример 6.13

Ниже показаны операции умножения и деления:

а. $g^9 \times g^{11} = g^{20} = g^{20 \bmod 15} = g^5 = g^2 + g \rightarrow (0110)$

б. $g^3 / g^8 = g^3 \times g^7 = g^{10} = g^2 + g + 1 \rightarrow (0111)$

Итоги раздела конечные поля

Конечное поле $GP(2^n)$ может использоваться для того, чтобы определить четыре операции — сложение, вычитание, умножение и деление n-битных слов. Только деление на нуль не определено. Каждое n-битовое слово может быть представлено как *полином* степени $n - 1$ с коэффициентами в $GF(2)$, — это означает, что операции на n-битовых словах могут быть представлены как операции на этом полиноме. При умножении двух полиномов необходимо сделать эти операции операциями по модулю. Для этого мы должны определить неприводимый *полином* степени n. Чтобы найти мультипликативные *инверсии* к полиномам, может быть применен расширенный *алгоритм Евклида*.

Рекомендованная литература

Нижеследующие книги и сайты обеспечивают более детальное рассмотрение понятий, обсужденных в этой лекции.

Книги

[Dur05], [Ros06J], [Bla03], [BW00] и [DF04] основательно рассматривают

алгебраические структуры.

Сайты

Нижеследующие сайты дают больше информации о темах, обсуждаемых в этой лекции.

- ссылка: http://en.wikipedia.org/wiki/Algebraic_structure - http://en.wikipedia.org/wiki/Algebraic_structure
- ссылка: http: // en.wikipedia.org/wiki/Ring _ % 28mathematics%29 - http://en.wikipedia.org/wiki/Ring_%28mathematics%29
- ссылка: http://en.wikipedia.org/wiki/Polynomials - http://en.wikipedia.org/wiki/Polynomials
- ссылка: http: // www.math.niu.edu / ~ rusin/known-math/index/20-XX.html - http://www.math.niu.edu/~rusin/known-math/index/20-XX.html
- ссылка: http: // www.math.niu.edu / ~ rusin/known-math/index/13-XX.html - http://www.math.niu.edu/~rusin/known-math/index/13-XX.html
- ссылка: http://www.hypermaths.org/quadibloc/math/abaint.htm - http://www.hypermaths.org/quadibloc/math/abaint.htm
- ссылка: http://en.wikipedia.org/wiki/Finite_field - http://en.wikipedia.org/wiki/Finite_field

Итоги

- Криптография требует заданных множеств и операций, определенных на этих множествах. Комбинации множеств и операций, приложенных к элементам этих множеств, есть алгебраическая структура. Были введены три алгебраических структуры: группы, кольца и поля.
- Группа — алгебраическая структура с *бинарной операцией*, удовлетворяющая четырем свойствам: замкнутость, ассоциативность, существование *тождества* (единичного элемента) и существование *инверсии*. Коммутативная группа, также называемая абелевой группой, — группа, оператор которой удовлетворяет дополнительному свойству: *коммутативности*.
- Подмножество `H` группы `G` — *подгруппа* `G`, если само `H` является группой с соответствующими операциями на `G`. Если *подгруппа*

группы может быть сгенерирована, используя степень элемента, *подгруппа* называется циклической *подгруппой*. Циклическая группа — это собственная циклическая *подгруппа* группы.

- Теорема Лагранжа связывает порядок группы и порядок ее *подгруппы*. Если порядок групп `G` и `H` соответственно `|G|` и `|H|`, тогда `|H|` делит `|G|`.
- Порядок элемента `a` в группе — наименьшее положительное целое число `n`, такое, что $a^n = e$ (единичному элементу).
- Кольцо — алгебраическая структура с двумя операциями. Первая операция должна удовлетворять всем пяти свойствам, требуемым для абелевой группы. Вторая операция должна удовлетворять только первым двум. Кроме того, вторая операция должна быть совместно с первой *дистрибутивной*. *Коммутативное* кольцо — это кольцо, в котором вторая операция удовлетворяет свойству *коммутативности*.
- Поле — *коммутативное* кольцо, в котором вторая операция удовлетворяет пяти свойствам, определенным для первой операции, за одним исключением: единичный элемент первой операции не имеет *инверсии*. Конечное поле, также называемое полем Галуа, — поле с элементами p^n, где `p` — простое число, а `n` — положительное целое число. GF(p^n) поля используется в операциях на n-битовых словах в криптографии.
- Для того чтобы представить `n` -битовые слова, используются полиномы с коэффициентами в `GF(2)`. Сложение и умножение `n` -битовых слов могут быть определены как сложение и умножение полиномов. Иногда проще определить элементы $GF(2^n)$ -поля, используя генератор. Если `g` — генератор поля, то $f(g) = 0$. Нахождение *инверсий* и выполнение операций на элементах поля становятся более простыми, когда элементы представлены как степени генератора поля.

Вопросы и упражнения

Обзорные вопросы

1. Определите алгебраическую структуру и назовите три

алгебраических структуры, обсужденные в этой лекции.

2. Определите группу и приведите различия между группой и *коммутативной* группой.
3. Определите кольцо и приведите различия между кольцом и *коммутативным* кольцом.
4. Определите поле и приведите различия между бесконечным полем и конечным полем.
5. Покажитеь число элементов в поле Галуа для простого числа.
6. Дайте один пример группы, использующей множество вычетов (операций по модулю).
7. Дайте один пример кольца, использующего множество вычетов (операций по модулю).
8. Дайте один пример поля, использующего множество вычетов (операций по модулю).
9. Покажите, как *полином* может представить n-битовое слово.
10. Определите неприводимый *полином*.

Упражнения

1. Для группы $G = <Z_4, +>$:
   - Докажите, что это — абелева группа.
   - Покажите результат операций $3 + 2$ и $3-2$ в этой группе.
2. Для группы $G =< Z_{6*}, \times >$:
   - Докажите, что это — абелева группа.
   - Покажите результат операций 5×1 и $1 \div 5$.
   - Покажите, почему мы не должны беспокоиться о делении на нуль в этой группе.
3. В таблице 5.1 для группы была определена только одна операция. Предположим, что эта операция — сложение. Покажите таблицу для операции вычитания (обратная операция).
4. Докажите, что перестановка в группе, показанной в таблице 5.2, не является *коммутативной*.
5. Докажите, что перестановка в группе, показанной в таблице 5.2, частично, в нескольких случаях, удовлетворяет свойству ассоциативности.
6. Создайте таблицу перестановки для двух входов и двух выходов, подобных показанным в таблице 5.2.

7. Алиса применяет три последовательных перестановки — `[1 3 2]`, `[3 2 1]` и `[2 1 3]`. Покажите, как Боб может использовать только одну перестановку, чтобы изменить процесс на противоположный. Пользуйтесь таблицей 5.2.
8. Найдите все *подгруппы* следующих групп:
   - $G =< Z_{16}, + >$
   - $G =< Z_{23}, + >$
   - $G =< Z_{16*}, \times >$
   - $G =< Z_{17*}, \times >$
9. Используя теорему Лагранжа, найдите порядок всех потенциальных подгрупп для следующих групп:
   - $G =< Z_{18}, + >$
   - $G =< Z_{29}, + >$
   - $G =< Z_{12*}, \times >$
   - $G =< Z_{19*}, \times >$
10. Найдите порядок всех элементов в следующих группах:
    - $G =< Z_{8}, + >$
    - $G =< Z_{7}, + >$
    - $G =< Z_{9*}, \times >$
    - $G =< Z_{7*}, \times >$
11. Повторите пример 6.12, используя неприводимый *полином* `f(x) = x^4 + x^3 +1`.
12. Повторите пример 6.13, используя неприводимый *полином* `f(x) = x^4 + x^3 +1`.
13. Повторите пример 6.12, используя неприводимый *полином* `f(x) = x^4 + x^3 +1`.
14. Какое выражение из нижеследующих является правильным полем Галуа?
    - `GF(12)`
    - `GF(13)`
    - `GF(16)`
    - `GF(17)`
15. Для каждого из следующих `n` -битовых слов найдите полиномы, которые представляют эти слова.
    - `10010`

- 10
- 100001
- 00011

16. Найдите n -битовое слово, которое представлено каждым из следующих полиномов:
    - $x^2 + 1$ в $GF(2^4)$
    - $x^2 + 1$ в $GF(2^5)$
    - $x + 1$ в $GF(2^3)$
    - x^7 в $GF(2^8)$
17. В поле GF(7) найдите результат следующих операций:
    - 5+3
    - 5−4
    - 5×3
    - $5 \div 3$
18. Докажите, что (x) и (x + 1) — неприводимые полиномы степени 1.
19. Докажите, что $(x^2 + x + 1)$ — неприводимый *полином* степени 2.
20. Докажите, что $(x^3 + x^2 + 1)$ — неприводимый *полином* степени 3.
21. Умножьте следующие 2 -битовые слова, используя полиномы:
    - (11) x (10)
    - (1010) x (1000)
    - (11100) x (10000)
22. Найдите мультипликативную *инверсию* следующих полиномов в $GF(2^2)$. (Заметим, что для этого поля есть только один модуль.)
    - 1
    - x
    - x + 1
23. Используйте расширенный евклидов алгоритм, чтобы найти *инверсию* $(x^4 + x^3 + 1)$ в $GF(2^5)$, применяя модуль $(x^5 + x^2 + 1)$.
24. Создайте таблицу сложения и умножения для $GF(2^4)$, используя $(x^4 + x^3 + 1)$ как модуль.
25. Используя таблицу 6.7, выполните следующие операции:

- $(100) \div (010)$
- $(100) \div (000)$
- $(101) \div (011)$
- $(000) \div (111)$

26. Покажите, как умножить $(x^3 + x^2 + x + 1)$ $(x^2 + 1)$ в $GF(2^4)$, используя алгоритм, приведенный в таблице 6.5, и пользуясь $(x^4 + x^3 + 1)$ как модулем.
27. Покажите, как умножить (10101) на (10000) в $GF(2^5)$, используя алгоритм, приведенный в таблице 6.5, и пользуясь $(x^5 + x^2 + 1)$ как модулем.

Введение в основы современных шифров с симметричным ключом

В этой лекции поставлено несколько целей. Показать различие между традиционными и современными шифрами с симметричным ключом. Привести современные блочные шифры и обсудить их характеристики. Объяснить, почему современные блочные шифры должны быть спроектированы как шифры подстановки. Ввести компоненты блочных шифров, таких как P-блоки и S-блоки. Обсудить и показать различие между двумя классами шифров: шифры Файстеля и шифры не-Файстеля. Обсудить два вида атак, особо направленных на раскрытие современных блочных шифров: дифференциальный и линейный криптоанализ. Ввести понятие "шифры для потока" и показать различие между синхронными и несинхронными шифрами. Обсудить линейную и нелинейную обратную связь регистров сдвига для реализации поточных шифров.

Традиционные шифры с *симметричным ключом*, которые мы изучали до сих пор, ориентируются на символы. С появлением компьютера стали необходимы шифры, ориентированные на бит. Потому что информация, которую надо зашифровать, — не всегда только текст; она может также состоять из чисел, графики, аудио- и видеоданных. Удобно преобразовать эти типы данных в поток битов, чтобы зашифровать этот поток, и затем передать зашифрованный поток. Кроме того, когда текст обработан на разрядном уровне, каждый символ заменен на `8` (или `16`) бит, а это означает, что число символов становится в `8` (или `16`) раз больше. Смешивание большего числа символов увеличивает безопасность.

Эта глава обеспечивает необходимую основу для изучения современных блочных и *поточных шифров*, которые рассматриваются в следующих трех главах. Большая часть этой главы посвящена обсуждению общих идей современных *блочных шифров*, и только малая часть — принципам современных *поточных шифров*.

7.1. Современные блочные шифры

Современный *блочный шифр* с симметричными ключами шифрует `n` -

битовый блок исходного текста или расшифровывает n -битовый блок зашифрованного текста. Алгоритм шифрования или дешифрования используют k -битовый ключ. Алгоритм дешифрования должен быть инверсией алгоритма шифрования, и оба в работе используют один и тот же ключ засекречивания так, чтобы Боб мог восстановить сообщение, передаваемое Алисой. Рисунок 7.1 показывает общую идею шифрования и дешифрования в современном блочном шифре.

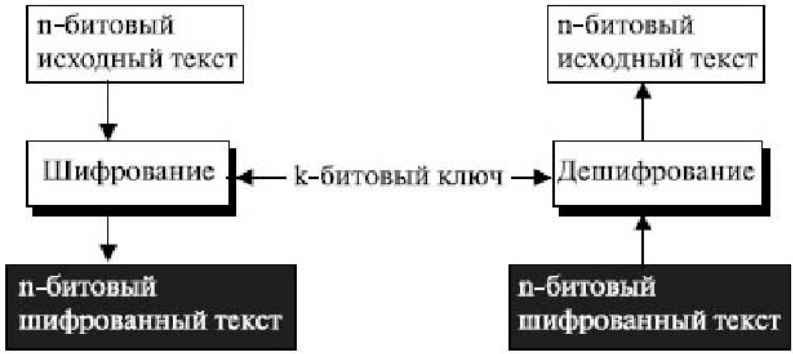


Рис. 7.1. Современный блочный шифр

Если сообщение имеет размер меньше, чем n бит, нужно добавить заполнение, чтобы создать этот n -разрядный блок; если сообщение имеет больше, чем n бит, оно должно быть разделено на n -разрядные блоки, и в случае необходимости нужно добавить к последнему блоку соответствующее заполнение. Общие значения для n обычно 64, 128, 256 или 512 битов.

Пример 7.1

Сколько дополнительных битов нужно добавить к сообщению 100 символов, если для кодирования используется ASCII по 8 битов и блочный шифр принимает блоки 64 бита?

Решение

Закодировать 100 символов, используя ASCII по 8 битов. Это сообщение содержит 800 бит. Исходный текст должен делиться без остатка на 64. Если | M | и | *Pad* | — длина сообщения и длина заполнения, то

| M | + | *Pad* | == 0 mod 64 -> | *Pad* | = -800 mod 64-> 32 mod 64

Это означает, что к сообщению нужно добавить 32 бита заполнения (например, нулей). Текст тогда будет состоять из 832 битов или тринадцати 64 -разрядных блоков. Заметим, что только последний блок содержит заполнение. *Щифратор* использует алгоритм шифрования тринадцать раз, чтобы создать тринадцать блоков зашифрованного текста.

Подстановка, или транспозиция

Современный *блочный шифр* может быть спроектирован так, чтобы действовать как шифр подстановки или как шифр транспозиции. Это — та же самая идея, которая используется и в традиционных шифрах, за исключением того, что символы, которые будут заменены или перемещены, содержат биты вместо символов.

Если шифр спроектирован как шифр подстановки, значения бита 1 или 0 в исходном тексте могут быть заменены либо на 0, либо на 1. Это означает, что исходный текст и зашифрованный текст могут иметь различное число единиц. Блок исходного текста на 64 бита, который содержит 12 нулей и 52 единицы, может быть представлен в зашифрованном тексте 34 нулями и 30 единицами. Если шифр спроектирован как шифр перестановки (транспозиции), биты только меняют порядок следования (перемещаются), сохраняя то же самое число символов в исходном и зашифрованном текстах. В любом случае, число возможных n -битовых исходных текстов или зашифрованных текстов равно 2^n, потому что каждый из n битов, использованных в блоке, может иметь одно из двух значений — 0 или 1.

Современные *блочные шифры* спроектированы как шифры подстановки, потому что свойства транспозиции (сохранение числа единиц или нулей) делают шифр уязвимым к атакам исчерпывающего поиска, как это показывают нижеследующие примеры.

Пример 7.2

Предположим, что мы имеем *блочный шифр*, где `n = 64`. Если есть `10` единиц в зашифрованном тексте, сколько испытаний типа "проб и ошибок" должна сделать Ева, чтобы получить исходный текст перехваченного зашифрованного текста в каждом из следующих случаев?

a. Шифр спроектирован как шифр подстановки.

b. Шифр спроектирован как шифр транспозиции.

Решение

a. В первом случае (подстановка) Ева понятия не имеет, сколько единиц находится в исходном тексте. Ева должна попробовать все возможные 2^{64} блока по `64` бита, чтобы найти один, который имеет смысл. Если бы Ева могла пробовать `1` миллиард блоков в секунду, и тогда ей потребовалось бы сотни лет, прежде чем эта работа могла бы принести успех.

b. Во втором случае (перестановка) Ева знает, что в исходном тексте есть точно `10` единиц, потому что транспозиция не изменяет числа единиц (или нулей) в зашифрованном тексте. Ева может начать атаку исчерпывающего поиска, используя только те `64` -битовые блоки, которые имеют точно `10` единиц. Есть только `(64!) / [(10!) (54!)] = 151 473 214 816` из 2^{64} слов по `64` бита, которые имеют точно `10` единиц. Ева может проверить всех их меньше чем за `3` минуты, если она может провести `1` миллиард испытаний в секунду.

Стойкий к атаке исчерпывающего поиска, современный блочный шифр должен быть спроектирован как шифр подстановки.

Блочные шифры как групповые математические перестановки

Как мы увидим в следующих лекциях, нам надо знать, является ли современный *блочный шифр* математической группой (см. лекции 5-6). Чтобы ответить на этот вопрос, сначала предположим, что ключ достаточно длинный, чтобы создать отображение любой возможной

входной информации в выходную. Он называется полноразмерным ключевым шифром. Практически, ключ меньше; длинный ключ можно применять только для некоторых отображений входной информации в выходную. Хотя *блочный шифр* должен иметь ключ, который является секретным при обмене между передатчиком и приемником, в шифре используются также компоненты, которые не зависят от ключа.

Полноразмерные ключевые шифры

Хотя полноразмерные ключевые шифры практически не используются, мы сначала обсудим их, чтобы сделать более понятным обсуждение шифров с ключом частичного размера.

Полноразмерные ключевые блочные шифры транспозиции. Такой ключевой шифр перемещает биты, не изменяя их значения, так что может быть смоделирован как перестановка n -мерного объекта с множеством $n!$ таблиц перестановки, в которых ключ определяет, какая таблица используется Алисой и Бобом. Мы должны иметь $n!$ *возможных ключей*, и такой ключ должен иметь длину $\lceil \log_2 n! \rceil$ бит.

Пример 7.3

Покажите модели и множество таблиц перестановки для *блочного шифра* транспозиции на 3 бита, где размер блока — 3 бита.

Решение

Множество таблиц перестановки имеет $3! = 6$ элементов, как показано на рис. 7.2. Ключ должен быть длиной $\lceil \log_2 n! \rceil = 3$ бита. Заметим, что хотя ключ на 3 бита может выбрать $2^3 = 8$ различных отображений, мы используем только 6 из них.

Рис. 7.2. Блочный шифр транспозиции в виде перестановки

Полноразмерные ключевые блочные шифры подстановки.Такие шифры не перемещают биты — они заменяют биты. На первый взгляд кажется, что полноразмерный ключевой шифр подстановки не может быть смоделирован как перестановка. Однако мы можем применить модель перестановки для шифра подстановки, если сможем декодировать *входную информацию* и кодировать выходную. Декодирование здесь означает преобразование n -разрядного целого числа в строку 2^n -бит с единственной единицей 1 и 2^n-1 нулями. Позиция единственной единицы указывает значение целого числа в упорядоченной последовательности позиций строки от 0 до $2^n - 1$. Поскольку новая входная информация имеет всегда единственную единицу, шифр может быть смоделирован как перестановка $2^n!$ объектов.

Пример 7.4

Покажите модель и множество таблиц перестановки для шифра подстановки блока на 3 бита.

Решение

Три входных исходных текста могут быть обозначены целыми числами от 0 до 7. Это может быть закодировано как строка, содержащая 8 битов с единственной единицей. Например, комбинация 000 может быть закодирована как 00000001 (первая единица справа); комбинация 101 может быть закодирована как 00100000 (шестая единица справа).

Рисунок 7.3 показывает модель и множество таблиц перестановки. Заметим, что число элементов в закодированном множестве намного больше, чем число элементов в шифре транспозиции (8! = 40 320). Ключ — также намного более длинный $\lceil \log_2 40320 \rceil = 16$ бит. Хотя ключ на 16 битов может определить 65 536 различных отображений, используются только 40 320.

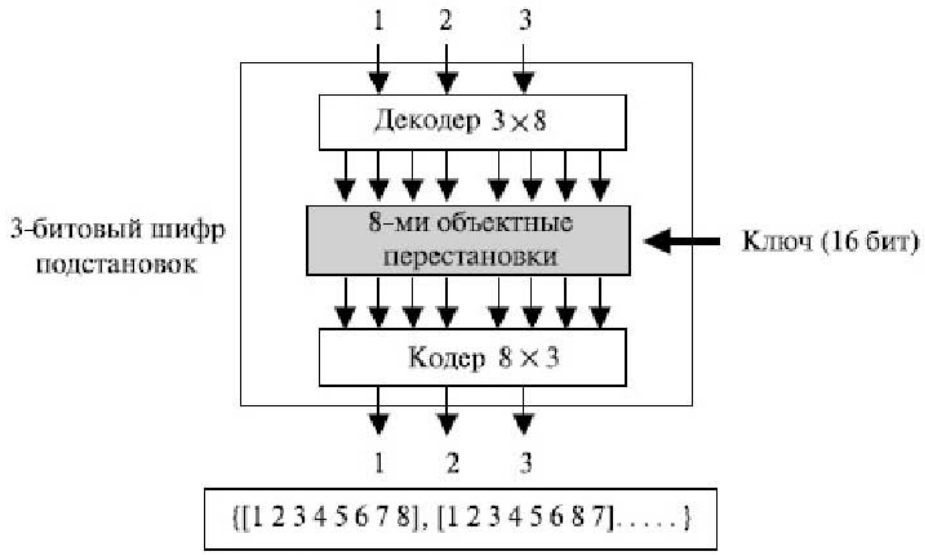


Рис. 7.3. Блочный шифр подстановки моделируется как шифр перестановки

Полноразмерный ключ — это n -разрядный шифр транспозиции или блочный шифр подстановки. Они могут быть смоделированы как шифры перестановки, но размеры их ключа различны: для шифра транспозиции ключ длиной — $[\log_2 n!]$ для шифра подстановки ключ длиной — $[\log_2 (2^n)!]$.

Групповая перестановка. Факт, что полноразмерная ключевая транспозиция или шифр подстановки/перестановки показывает, что если шифрование (или дешифрование) использует больше чем одну любую комбинацию из этих шифров, результат эквивалентен операции групповой перестановки. Как уже обсуждалось в лекциях 5-6, две или

больше каскадных перестановки могут всегда быть заменены единственной перестановкой. Это означает, что бесполезно иметь $2^{2^{70}}$ больше чем один каскад полноразмерных ключевых шифров, потому что эффект тот же самый, как и при наличии единственного шага.

Шифры ключа частичного размера

Фактические шифры не могут использовать полноразмерные ключи, потому что размер ключа становится несуразно большим, особенно для *блочного шифра* подстановки. Например, общий шифр подстановки – *DES* — (см. лекцию 11) применяет 64 -разрядный *блочный шифр*. Если бы проектировщики *DES* использовали полноразмерный ключ, он был бы $\log_2\left(2^{64}!\right) = 2^{70}$ битов. На практике ключ для *DES* — только 56 битов, что является очень маленьким фрагментом полноразмерного ключа. Это означает, что *DES* использует только 2^{56} отображений из приблизительно $2^{2^{70}}$ возможных отображений.

Группа перестановки. Зададим себе вопрос: можно ли установить, что многоступенчатая транспозиция с *частичным ключом* или подстановка — это группа перестановки с композицией операций? Ответ на этот вопрос чрезвычайно важен, потому что он говорит нам о том, является ли многоступенчатая версия с частичным шифром таким же средством шифрования, как и сам шифр. Этот факт позволяет достигнуть большей степени безопасности (см. обсуждение многократной *DES* в лекции 11).

Частичный ключевой шифр – это группа, если это — подгруппа соответствующего размера ключа шифра. Другими словами, если полноразмерный ключевой шифр — это группа G = <M, ○>, где M. — множество отображений и (○) — композиция операций, то шифр с ключом частичного размера должен представлять *подгруппу* H = <N, ○>, где N — подмножество M с теми же самыми операциями.

Например, было доказано, что многоступенчатый *DES* с 56 -битовым ключом не является группой, потому что подгруппа с 2^{56} отображениями не может быть создана из группы с 2^{64}! отображениями.

Частичный ключевой шифр есть группа с набором операций, если он является подгруппой соответствующего полноразмерного ключевого шифра.

Шифры без ключа

Хотя использование отдельно шифра без ключа фактически бесполезно, возможно их применение в качестве компонентов ключевых шифров,

Шифр транспозиции без ключа. Шифры без ключа (или с фиксированным ключом) можно рассматривать как шифр транспозиции, реализованный в аппаратных средствах. Фиксированный ключ (единственное правило перестановки) может быть представлен как таблица в случае реализации шифра в программном обеспечении. Следующая часть этой лекции обсуждает шифры транспозиции без ключа, названные P-блоками, которые используются как стандартные блоки современных *блочных шифров*.

Шифры подстановки без ключа.Такой шифр без ключа (или с фиксированным ключом) можно представить себе как заранее определенное отображение входной информации к выходной. Отображение может быть представлено как таблица, как *математическая функция*, а также другими способами. В следующей части этой лекции рассматриваются шифры подстановки без ключей, названные S -блоками, которые применяются как стандартные блоки современных *блочных шифров*.

Компоненты современного блочного шифра

Современные *блочные шифры* обычно являются ключевыми шифрами подстановки, в которых ключ позволяет только частичные отображения возможных входов информации в возможные выходы. Однако эти шифры обычно не проектируются как единый модуль. Чтобы обеспечивать требуемые свойства современного *блочного шифра*, такие как рассеяние и перемешивание информации (обсуждается кратко), этот шифр формируется как комбинация модулей транспозиции (называемых P -блоками), модулей подстановки (называемых S -блоками) и

некоторыми другими модулями (обсуждается кратко).

P -блок (блок перестановки) подобен традиционному шифру транспозиции символов. Он перемещает биты. В современных *блочных шифрах* мы можем найти три типа P -блоков: прямые P -блоки, P -блоки расширения и P -блоки сжатия, что и показано на рис. 7.4.

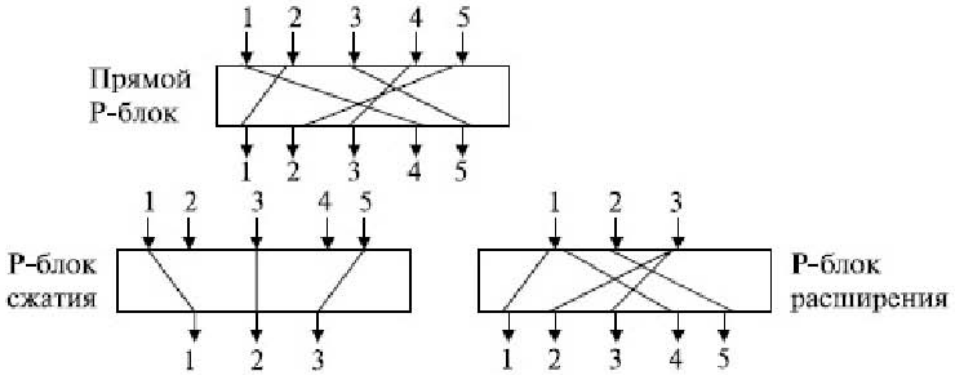


Рис. 7.4. Три типа P-блоков

Рисунок 7.4 показывает прямой P -блок 5×5, P -блок сжатия 5×3 и P -блок расширения 3×5. Рассмотрим каждый из них более подробно.

Прямые P-блоки.Прямой P -блок с n входами и n выходами – это перестановка с n! возможными отображениями.

Пример 7.5

Рисунок 7.5 показывает все 6 возможных отображений P -блока 3×3.

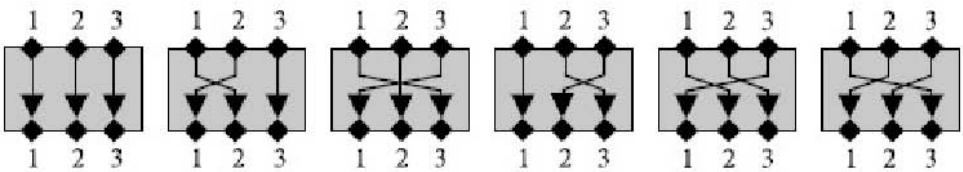


Рис. 7.5. Возможные отображения P-блока 3x3

Хотя P -блок может использовать ключ, чтобы определить одно из n!

отображений, обычно P -блоки – без применения ключа, то есть отображение задано заранее. Если P -блок задан заранее и замонтирован в аппаратных средствах или если он реализован в программном обеспечении, таблицы перестановок задают правило отображения. Во втором случае входы в таблице указывают в позиции, в которых указаны позиции выходов. Таблица 7.1 дает пример таблицы перестановок, когда n равно 64.

Таблица 7.1. Пример таблицы перестановки для прямого P-блока

58	50	42	34	26	18	10	02	60	52	44	36	28	20	12	04
62	54	46	38	30	22	14	06	64	56	48	40	32	24	16	08
57	49	41	33	25	17	09	01	59	51	43	35	27	19	11	03
61	53	45	37	29	21	13	05	63	55	47	39	31	23	15	07

Таблица 7.1 имеет 64 табличных входа, которые фиксируют соответствие 64 информационным входам. Позиция (индекс) входа соответствует выходу. Например, первый табличный вход содержит номер 58. Это означает, что первый выход будет соответствовать 58 -му входу. Поскольку последний табличный вход — 7, это означает, что, 64 -й выход будет соответствовать седьмому информационному входу, и так далее.

Пример 7.6

Составьте таблицу перестановки для прямого P -блока 8 x 8, которая перемещает два средних бита (биты 4 и 5) во входном слове к двум крайним битам (биты 1 и 8) выходного слова. Относительные позиции других битов не изменяются.

Решение

Нам надо создать прямой P -блок с таблицей [4 1 2 3 6 7 8 5]. Относительные позиции бит 1, 2, 3, 6, 7 и 8 не меняются, но первый информационный выход связан с четвертым информационным входом, восьмой информационный выход — с пятым информационным входом.

P-блоки сжатия. P -блок сжатия – это P -блок с n входами и m выходами, где m <n. Некоторые из информационных входов блокированы и не связаны с выходом (см. рисунок 7.4). P -блоки сжатия, используемые в современных *блочных шифрах*, обычно являются безключевыми с таблицей перестановки, которая указывает правила перестановки бит. Нам надо учитывать, что таблица перестановок для P -блока сжатия имеет m табличных входов, но в содержании каждого табличного входа – от 1 до n, и некоторые из них могут отсутствовать (те информационные входы, которые блокированы). Таблица 7.2 показывает пример таблицы перестановки для P -блока сжатия 32×24. Обратите внимание, что входы 7, 8, 9, 16, 23, 24 и 25 блокированы.

Таблица 7.2. Пример таблицы перестановки 32x24

01	02	03	21	22	26	27	28	29	13	14	17
18	19	20	04	05	06	10	11	12	30	31	32

P -блоки сжатия используются, когда мы должны переставить биты и в то же время уменьшить число битов для следующей ступени.

P-блок расширения — P -блок с n входами и m выходами, где m> n. Некоторые из входов связаны больше чем с одним выходом (см. рис. 7.4). P -блоки расширения, используемые в современных *блочных шифрах*, обычно без ключа. Правила перестановки бит указываются в таблице. Таблица перестановки для P -блока расширения имеет m табличных входов, но m – n входов (те входы, которые связаны больше чем с одним информационным выходом). Таблица 7.3 показывает пример таблицы перестановки для P -блока расширения 12 16. Обратите внимание, что каждый из 1, 3, 9 и 12 соединен с двумя выходами.

Таблица 7.3. Пример таблиц перестановки 12x16

01	09	10	11	12	01	02	03	03	04	05	06	07	08	09	12

P -блоки расширения используются, когда мы должны переставить

биты и то же время увеличить число битов для следующего каскада шифрования.

Обратимость.Прямой P -блок является обратимым. Это означает, что мы можем использовать прямой P -битовый шифр и дешифровать его. Таблицы перестановки, однако, должны быть обратимыми по отношению друг к другу. В лекциях 5-6 мы видели, как можно получить обратную таблицу перестановки.

Пример 7.7

Рисунок 7.6 показывает, как изменить таблицу перестановки в случае одномерной таблицы.

Рис. 7.6. Изменение таблицы перестановки

P -блоки сжатия и расширения *необратимы*. В P -блоках сжатия вход может быть отброшен в процессе шифрования; алгоритм дешифрования не имеет ключа, чтобы восстановить отброшенный бит. В P -блоке расширения вход в процессе шифрования может быть отображен более чем в один *выход; алгоритм* дешифрования не имеет ключа и не определяет тем самым, какие из нескольких входов отображены в данном выходе. Рисунок 7.7 демонстрирует оба случая.

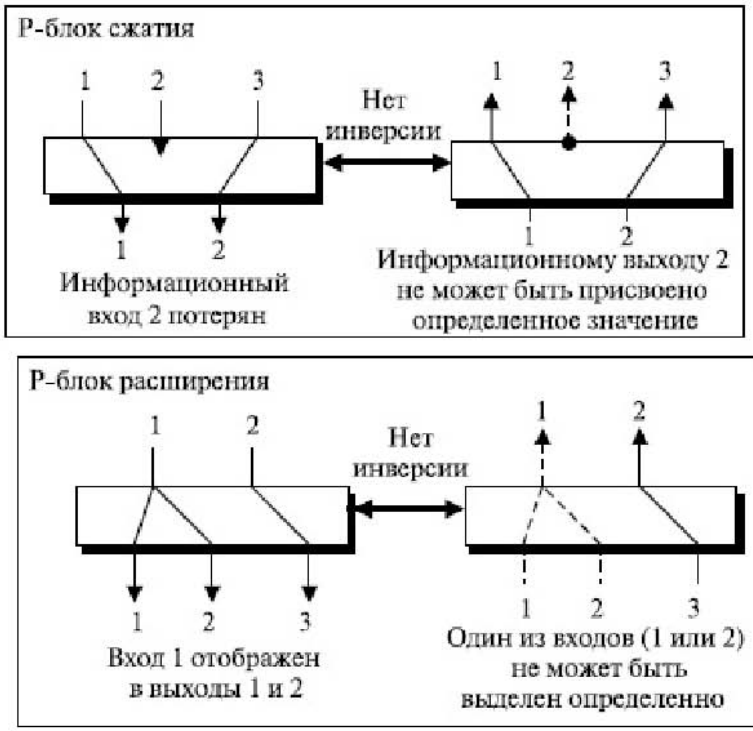


Рис. 7.7. P-блоки сжатия и расширения как необратимые компоненты

Рисунок 7.7 также показывает, что P -блок сжатия не является обратным шифром P -блока расширения и наоборот. Это означает, что если мы используем P -блок сжатия для шифрования, мы не сможем использовать P -блок расширения для дешифрования и наоборот. Однако, как будет показано позже в этой лекции, есть шифры, которые применяют P -блоки сжатия или расширения для шифрования; но их эффективность хуже, чем у некоторых других способов.

Прямой P -блок является обратимым, а P -блоки сжатия и расширения – нет.

S-блоки

S-блок (блок подстановки) можно представить себе как миниатюрный шифр подстановки. Этот блок может иметь различное число входов и

выходов. Другими словами, вход к S -блоку может быть n -битовым словом, а выход может быть m разрядным словом, где m и n — не обязательно одинаковые числа. Хотя S -блок может быть ключевым или без ключа, современные *блочные шифры* обычно используют S -блоки без ключей, где отображение от информационных входов к информационным выходам заранее определено.

S -блок — m x n модуль подстановки, где m и n не обязательно равны.

Линейный и нелинейный S-блоки. В S -блоке с n входами и m выходами мы обозначим входы x_0, x_1, …., x_n и выходы y_1 , . . . , y_m. Соотношения между входами и выходами могут быть представлены как система уравнений

$y_1 = f_1 (x_1, x_2, …, x_n)$

$y_2 = f_2 (x_1, x_2, …, x_n)$

….

$y_m = f_m (x_1, x_2, …, x_n)$

В линейном S-блоке вышеупомянутые соотношения могут быть выражены как

$$y_1 = a_{1,1}x_1 \oplus a_{1,2}x_2 \oplus \cdots \oplus a_{1,n}x_n$$
$$y_2 = a_{2,1}x_1 \oplus a_{2,2}x_2 \oplus \cdots \oplus a_{2,n}x_n$$
$$\cdots$$
$$y_m = a_{m,1}x_1 \oplus a_{m,2}x_2 \oplus \cdots \oplus a_{m,n}x_n$$

В нелинейном S-блоке мы не можем всегда задать для каждого выхода указанные выше соотношения.

Пример 7.8

В S -блоке с тремя входами и двумя выходами мы имеем

$$y_1 = x_1 \oplus x_2 \oplus x_3 y_2 = x_1$$

S -блок линеен, потому что $a_{1,1} = a_{1,2} = a_{1,3} = a_{2,1}=1$ и $a_{2,2} = a_{2,3} = 0$. Эти соотношения могут быть представлены матрицами, как показано ниже:

$$\begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \begin{pmatrix} 111 \\ 100 \end{pmatrix} \times \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$$

Пример 7.9

В S -блоке с тремя входами и двумя выходами мы имеем

$$y_1 = (x_1)^3 + x_2 \quad y_2 = (x_1) + x_1x_2 + x_3$$

где умножение и сложение проводится в GF(2). S -блок нелинеен, потому что нет линейных соотношений между входами и выходами.

Пример 7.10

Следующая таблица определяет отношения между входами/выходами для S -блока размера 3×2. Крайний левый бит входа определяет строку; два самых правых бита входа определяют столбец. Два бита выхода – это значение на пересечении секции выбранной строки и столбца.

Самый левый бит

	00	01	10	11
0	00	10	01	11
1	10	00	11	01

← Самые правые биты

Биты выходов

Обратимость. S -блоки — шифры подстановки, в которых отношения между входом и выходом определены таблицей или математическим соотношением. S -блок может быть или может не быть обратимым. В обратимом S -блоке число входных битов должно быть равным числу бит выхода.

Пример 7.11

Рисунок 7.8 показывает пример обратимого S -блока. Одна из таблиц используется в алгоритме шифрования; другая таблица — в алгоритме дешифрования. В каждой таблице крайний левый бит входа определяет строку; следующие два бита определяют столбец. Выход — это значение на пересечении строки и столбца.

Например, если вход к левому блоку — 001, выход — 101. Вход 101 в правой таблице дает выход 001. Это показывает, что эти две таблицы позволяют получить обратный результат по отношению друг к другу.

Исключающее или

Важный компонент в большинстве блоков шифрования — операция ИСКЛЮЧАЮЩЕЕ ИЛИ: Как мы уже обсуждали в лекции 7, операции сложения и вычитания в $GF(2^n)$ выполняется с помощью одной и той же операции, называемой ИСКЛЮЧАЮЩЕЕ ИЛИ или (XOR):

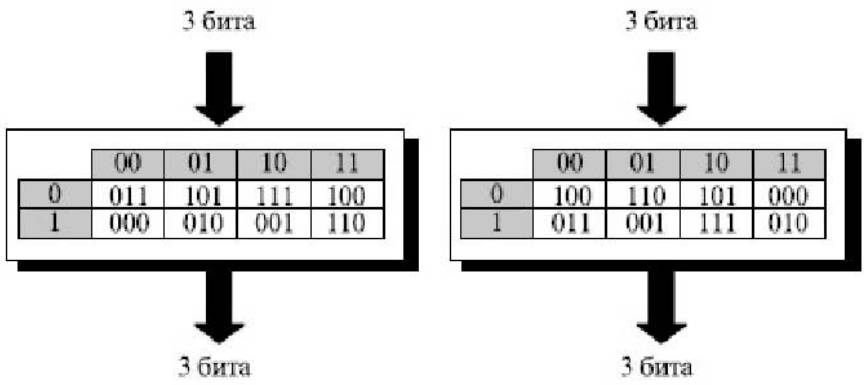


Рис. 7.8. Таблицы S-блока для примера 7.11

Свойства. Пять свойств операции ИСКЛЮЧАЮЩЕЕ ИЛИ в поле $GF(2^n)$ делают эту операцию очень удобной для использования в блочном шифре.

1. Замкнутость. Это свойство гарантирует, что в результате этой операции два n -битовых слова дают другое n -битовое слово.

2. Ассоциативность. Это свойство позволяет нам использовать больше

чем одно ИСКЛЮЧАЮЩЕЕ ИЛИ, которые можно вычислять в любом порядке.

$$x \oplus (y \oplus z) \leftrightarrow (x \oplus y) \oplus z$$

3. Коммутативность. Это свойство позволяет нам менять местами операторы (*входную информацию*), не изменяя результат (*выходную информацию*).

$$x \oplus y \leftrightarrow y \oplus x$$

4. Существование нулевого (тождественного) элемента. Нулевой элемент для операции ИСКЛЮЧАЮЩЕЕ ИЛИ – слово, которое состоит из всех нулей, или `(00... 0)`. Это подразумевает, что существует слово с нейтральными элементами, которое при проведении операции не изменяет слово.

$$x \oplus (00 \ldots .00) = x$$

Мы используем это свойство в шифре Файстеля, который рассмотрим позже в этой лекции.

5. Существование инверсии. В поле $GF(2^n)$ каждое слово есть аддитивная инверсия самого себя. Это подразумевает, что проведение операции ИСКЛЮЧАЮЩЕЕ ИЛИ слова с самим собой приводит к нулевому элементу:

$$x \oplus x = (00 \ldots 0)$$

Мы также используем это свойство в шифре Файстеля, который рассмотрим позже в этой лекции.

Дополнение. *Операция дополнения — одноместная операция* (один информационный вход и один информационный выход), которая инвертирует каждый бит в слове. `0` -вой бит меняет на `1` (единичный) бит; `1` (единичный) бит меняет на `0` -вой бит. Нас интересует операции с дополнением относительно операции ИСКЛЮЧАЮЩЕЕ ИЛИ. Если $\bar{x}$ — дополнение "`x`", тогда верны следующие два соотношения:

$x \oplus \bar{x} = (11 \ldots 1)$ и $x \oplus (11 \ldots 1) = \bar{x}$

Мы также используем эти свойства позже в этой лекции, когда будем обсуждать безопасность некоторых шифров.

Инверсия. Инверсия компонента в шифре имеет смысл, если компонент представляет одноместную операцию (один вход и один выход). Например, `P` -блок без ключа или `S` -блок без ключа могут быть обратимыми, потому что они имеют один вход и один выход. Операция ИСКЛЮЧАЮЩЕЕ ИЛИ — *бинарная операция*. Инверсия операции ИСКЛЮЧАЮЩЕЕ ИЛИ может иметь смысл, только если один из входов зафиксирован (один и тот же при шифровании и дешифровании). Например, если один из входов — ключ, который обычно является одним и тем же в шифровании и дешифровании, тогда операция ИСКЛЮЧАЮЩЕЕ ИЛИ является обратимой, как показано на рис. 7.9.

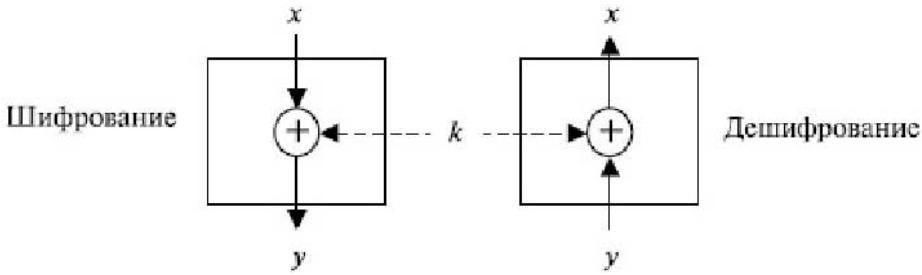


Рис. 7.9. Обратимость операции ИСКЛЮЧАЮЩЕЕ ИЛИ

На рисунке 7.9 свойство аддитивной инверсии подразумевает, что

$$y = x \oplus kx = k \oplus y$$

Мы используем это свойство, когда позже в этой лекции будем обсуждать структуру *блочных шифров*.

Циклический сдвиг

Другой компонент, применяемый в некоторых современных *блочных шифрах*, – операция циклического сдвига. Смещение может быть влево или вправо. Круговая операция левого сдвига сдвигает каждый бит в `n`

-битовом слове на k позиции влево; крайние левые k -биты удаляются слева и становятся самыми правыми битами. Круговая операция правого сдвига сдвигает каждый бит в n -битовом слове на k позиций вправо; самые правые k -биты справа удаляются и становятся крайними левыми битами. Рисунок 7.10 показывает и левые и правые операции в случае, где n = 8 и k = 3.

Циклическая *операция сдвига* смешивает биты в слове и помогает скрыть образцы в первоначальном слове. Хотя число позиций, на которые биты будут сдвинуты, может использоваться как ключ, циклическая *операция сдвига* обычно – без ключа; значение k устанавливается и задается заранее.

Обратимость. Циклическая операция левого сдвига – инверсия операции правого сдвига. Если одна из них используется для шифрования, другая может применяться для дешифрования.

Свойства. *Операция циклического сдвига* имеет два свойства, которые нам надо знать.

Первая — это смещение по модулю n. Другими словами, если k = 0 или k = n, никакого смещения не происходит. Если k является большим, чем n, тогда входная информация сдвинута на k mod n бит. Второе свойство, *операция циклического сдвига* над соединением операций - есть группа. Это означает, что если смещение делается неоднократно, то одно и то же значение может появиться несколько раз..

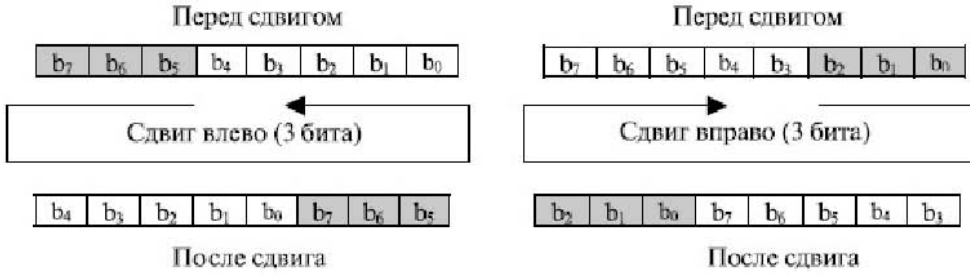


Рис. 7.10. Циклический сдвига 8 битового слова налево или направо

Замена

Операция замены — специальный случай операции циклического сдвига, где `k = n/2` означает, что эта операция возможна, только если `n` — четный номер. Поскольку сдвиг влево `n/2` — то же самое, что сдвиг `n/2` вправо, эта операция является обратимой. Операция замены для шифрования может быть полностью раскрыта операцией замены для дешифрации. Рисунок 7.11 иллюстрируетт операцию замены для слова на `8` битов.

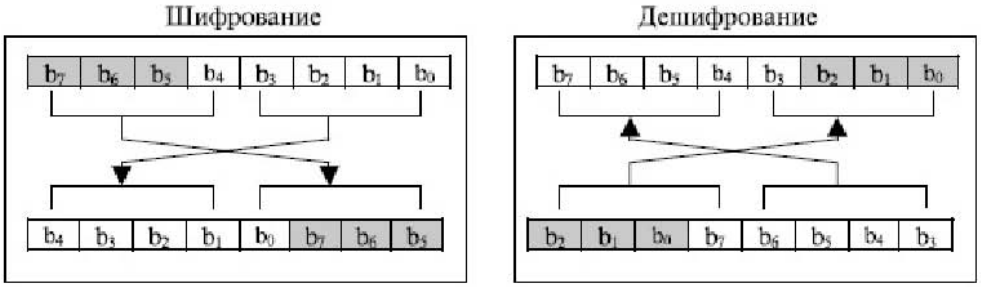


Рис. 7.11. Операция замена в 8 битовом а слове

Разбиение и объединение

Две других операции, применяемые в некоторых *блочных шифрах*, — разбиение и объединение. Разбиение обычно разделяет `n` -битовое слово в середине, создавая два слова равной длины. Объединение связывает два слова равной длины, чтобы создать `n` -битовое слово. Эти две операции инверсны друг другу и могут использоваться как пара, чтобы уравновесить друг друга. Если одна используется для шифрования, то другая — для дешифрования. Рисунок 7.12 показывает эти две операции для случая `n = 8`.

Составные шифры

Шеннон ввел понятие составные шифры. Составной шифр – комплекс, который объединяет подстановку, перестановку и другие компоненты, рассмотренные в предыдущих разделах.

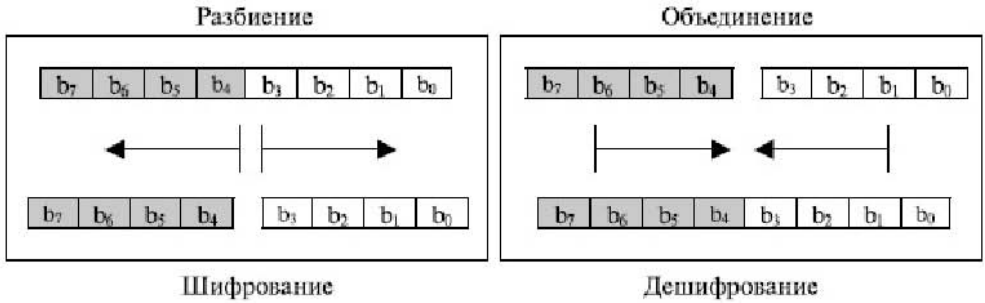


Рис. 7.12. Операции разбиение и объединения с 8 - битовым словом

Рассеивание и перемешивание

Идея Шеннона в представлении составного шифра должна была дать возможность блочным шифрам иметь две важных свойства: рассеяние и перемешивание. Рассеивание должно скрыть отношения между зашифрованным текстом и исходным текстом. Это собьет с толку противника, который использует статистику зашифрованного текста, чтобы найти исходный текст. Рассеивание подразумевает, что каждый символ (символ или бит) в зашифрованном тексте зависит от одного или всех символов в исходном тексте. Другими словами, если единственный символ в исходном тексте изменен, несколько или все символы в зашифрованном тексте будут также изменены.

Рассеивание скрывает отношения между зашифрованным текстом и исходным текстом.

Идея относительно перемешивания — в том, что оно должно скрыть отношения между зашифрованным текстом и ключом. Это собьет с толку противника, который стремится использовать зашифрованный текст, чтобы найти ключ. Другими словами, если единственный бит в ключе изменен, все биты в зашифрованном тексте будут также изменены.

Перемешивание скрывает отношения между зашифрованным текстом и ключом.

Раунды

Распыление и перемешивание могут быть достигнуты использованием повторения составных шифров, где каждая итерация — комбинация S -блоков, P -блоков и других компонентов. Каждая итерация называется раундом.*Блочный шифр* использует ключевой список,или генератор ключей, который создает различные ключи для каждого раунда от ключа шифра. В N -раундном шифре, чтобы создать зашифрованный текст, исходный текст шифруется N раз; соответственно, зашифрованный текст расшифровывается N раз. Текст, созданный на промежуточных уровнях (между двумя раундами), называется средним текстом.Рисунок 7.13 показывает простой составной шифр с двумя раундами. На практике составные шифры имеют больше чем два раунда. На рис. 7.13 в каждом раунде проводятся три преобразования:

а. 8 -битовый текст смешивается с ключом, чтобы сделать символы текста равновероятными (скрыть биты, используя ключ) — "отбелить" текст (whiting). Это обычно делается с помощью операции ИСКЛЮЧАЮЩЕЕ ИЛИ слова на 8 битов с ключом на 8 битов.

б. Выходы "отбеливателя" разбиты на четыре группы по 2 бита и подаются в четыре S -блока. Значения битов изменяются в соответствии с построением S -блоков в этом преобразовании.

с. Выходы S -блоков поступают в P -блок, при этом биты переставлены так, чтобы в следующем раунде результат каждого блока поступил на различные входы.

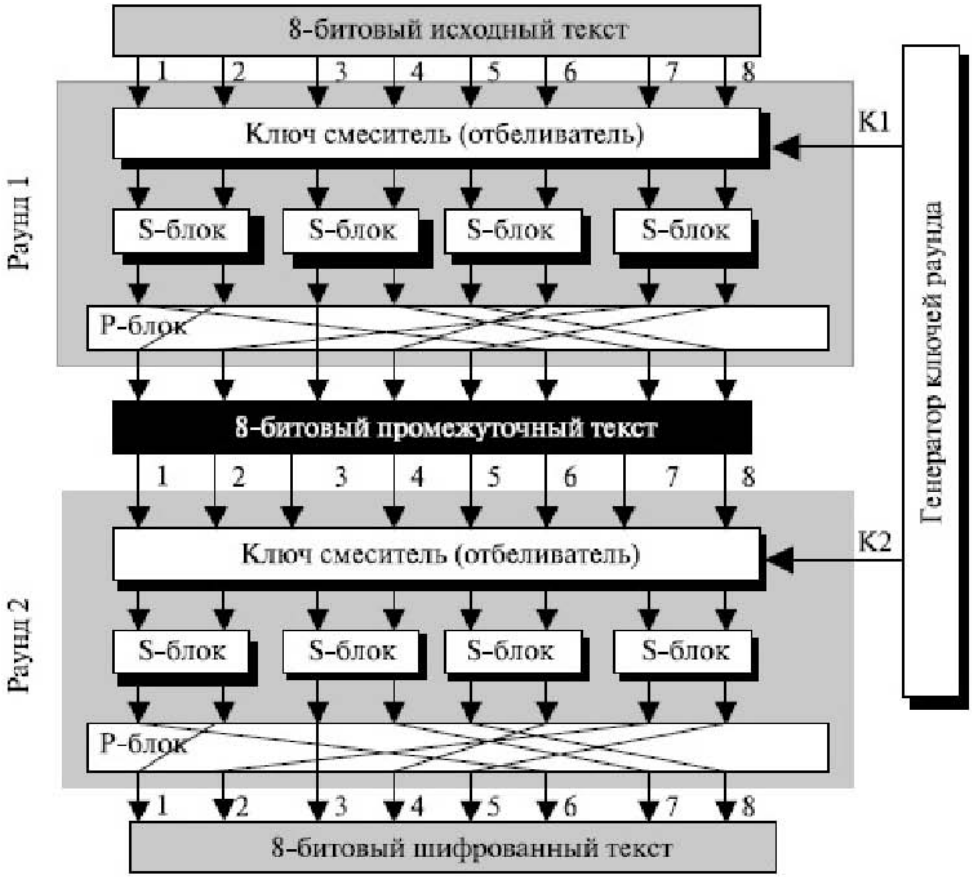


Рис. 7.13. Составнной шифр, состоящий из двух раундов

Рассеивание, которое показано на упрощенном рис. 7.13 как составной шифр, используя комбинацию S -блоков и P -блоков, может гарантировать рассеивание.

а. В первом раунде бит 8, после проведения операции ИСКЛЮЧАЮЩЕЕ ИЛИ с соответствующими битами ключа K_1, изменяет два бита (биты 7 и 8) через S -блок 4. Бит 7 переставлен и становится битом 2 ; бит 8 переставлен и становится битом 4. После первого раунда бит 8 изменяет биты 2 и 4. Во втором раунде бит 2 после проведения операции ИСКЛЮЧАЮЩЕЕ ИЛИ с соответствующими битами ключа K_2 изменяет два бита (биты 1 и 2) через S -блок 1. Бит 1 – переставлен и становится битом 6 ; бит 2

переставлен и становится битом 1. Бит 4 после проведения операции ИСКЛЮЧАЮЩЕЕ ИЛИ с соответствующим битом в K_2 изменяет биты 3 и 4. Бит 3 остается, бит 4 переставлен и становится битом 7. После второго раунда из 8 бит изменены биты 1, 3, 6 и 7.

b. Прохождение этих шагов в другом направлении (от зашифрованного текста до исходного текста) показывает, что каждый бит в зашифрованном тексте изменяет исходный текст на несколько битов.

Перемешивание. На рисунке 7.14 показано, как изменение единственного бита в исходном тексте вызывает изменение многих битов в зашифрованном тексте. Рисунок 7.14 также доказывает нам, что свойство перемешивания может быть получено с помощью составного шифра. Четыре бита зашифрованного текста, биты 1, 3, 6 и 7 преобразованы с помощью трех битов в ключах (бит 8 в K_1 и битах 2 и 4 в K_2). Прохождение в обратном направлении показывает, что каждый бит ключа в каждом раунде затрагивает несколько битов в зашифрованном тексте. Отношения между битами зашифрованного текста и ключевыми битами показаны в затененных прямоугольниках.

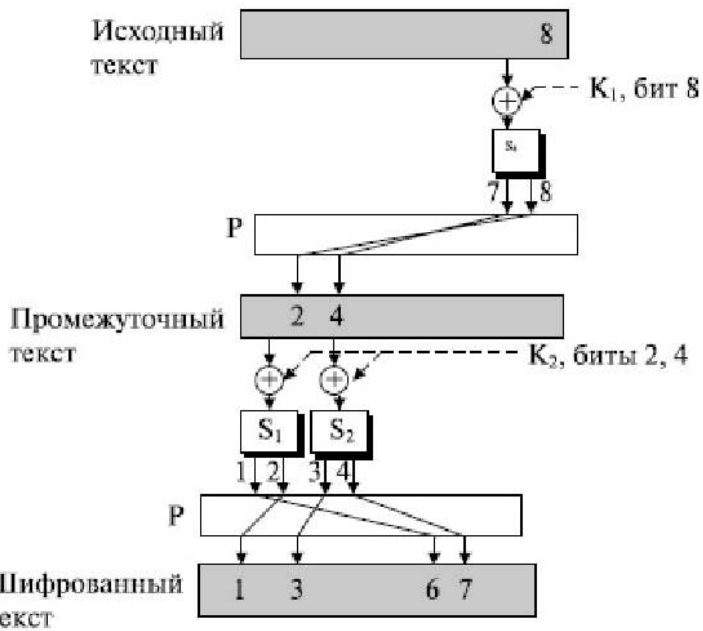


Рис. 7.14. Рассеивание и перемешивание в блочном шифре

Практические шифры. Чтобы улучшить рассеивание и перемешивание, практические шифры используют крупные блоки данных, больше S -блоков и больше раундов. Очевидно, что некоторое увеличение числа раундов при использовании большого числа S -блоков может создать лучший шифр, в котором зашифрованный текст выглядит все более как случайное n -битовое слово. Таким образом, отношения между зашифрованным текстом и исходным текстом будут полностью скрыты (рассеяны). Увеличение числа раундов увеличивает число ключей раундов, что лучше скрывает отношения между зашифрованным текстом и ключом.

Два класса составных шифров

Современные *блочные шифры* — все составные, но они разделены на два класса. Шифры в первом классе используют и обратимые, и *необратимые* компоненты. Эти шифры упоминаются обычно как шифры Файстеля. *Блочный шифр DES* (*DATA ENCRYPTION STANDARD*), обсуждаемый в лекции 11, — хороший пример шифра Файстеля. Шифры во втором классе применяют только обратимые компоненты. Обращаем ваше внимание на шифры в этом классе как шифры не-Файстеля (из-за отсутствия другого названия). *Блочный шифр AES* (ADVANCED ENCRYPTION STANDARD), обсуждаемый в лекциях 12-13, — хороший пример шифра не-Файстеля.

Шифры Файстеля

Файстель проектировал очень интеллектуальный и интересный шифр, который использовался в течение многих десятилетий. Шифр Файстеля может иметь три типа компонентов: самообратимый, обратимый и *необратимый*.

Шифр Файстеля содержит в блоках все *необратимые* элементы и использует один и тот же модуль в алгоритмах дешифрования и шифрования. Вопрос в том, как алгоритмы шифрования и дешифрования позволяют инвертировать открытый и закрытый тексты друг в друга, если каждый содержит *необратимый* модуль. Файстель показал, что они могут быть сбалансированы.

Первая идея. Чтобы лучше понять шифр Файстеля, давайте посмотрим, как мы можем использовать один и тот же *необратимый* компонент в алгоритмах дешифрования и шифрования. Эффекты *необратимого* компонента в алгоритме шифрования могут быть отменены в алгоритме дешифрования, если мы используем операцию ИСКЛЮЧАЮЩЕЕ ИЛИ, как показано на рис. 7.15.

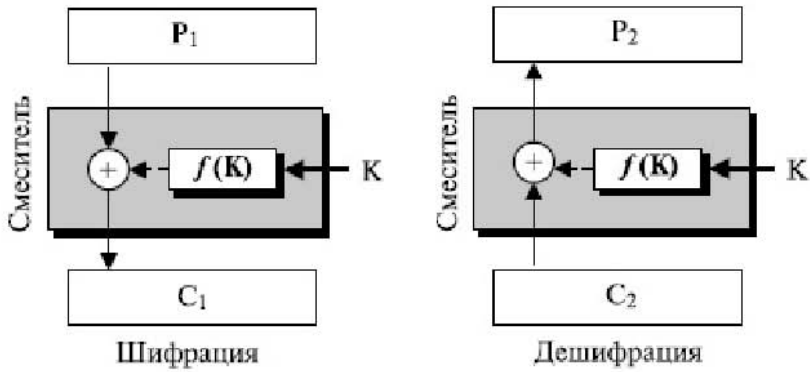


Рис. 7.15. Первая идея в разработке шифра Файстеля

В шифровании ключ поступает на вход необратимой функции `f (K)`, которая является одним из слагаемых оператора ИСКЛЮЧАЮЩЕГО ИЛИ с исходным текстом. Результат становится зашифрованным текстом. Мы будем называть комбинацию функции и операции ИСКЛЮЧАЮЩЕЕ ИЛИ смесителем (из-за отсутствия другого названия). Смеситель играет важную роль в более поздних вариантах шифра Файстеля.

Поскольку ключ один и тот же в шифровании и дешифровании, мы можем доказать, что два алгоритма инверсны друг другу. Другими словами, если $C_2 = C_1$ (любое изменение в зашифрованном тексте в течение передачи), то $P_2 = P_1$.

$$\text{Шифрование}: C_1 = P_1 \oplus f(K)$$
$$\text{Дешифрование}: P_2 = C_2 \oplus f(K) = C_1 \oplus f(K) = P_1 \oplus f(K) \oplus f(K) = P_1 \oplus (00\ldots0) = P_1$$

Обратите внимание, что использовались два свойства операции ИСКЛЮЧАЮЩЕЕ ИЛИ (существование инверсии и существование

нулевого кода).

Уравнения, показанные выше, доказывают, что хотя смеситель имеет неконвертируемый элемент, сам смеситель является самоконвертируемым.

Пример 7.12

Это тривиальный пример. Имеется исходный текст и зашифрованный текст, каждый 4 бита длиной, и ключ 3 бита длиной. Предположим, что функция извлекает первый и третий биты ключа, интерпретирует биты как десятичный номер, находит квадрат этого числа и интерпретирует результат как 4 -битовую двоичную последовательность. Покажите результаты шифрования и дешифрования, если первоначальный исходный текст — 0111, и ключ — 101.

Решение

Функция извлекает первые и третьи биты ключа и получается в результате 11 в двоичном виде или 3 в десятичном отображении. Результат возведения во вторую степень (квадрат) — 9, в двоичном отображении 1001.

$$\text{Шифрование}: C = P \oplus f(K) = 0111 \oplus 1001 = 1110$$
$$\text{Дешифрование}: P = C \oplus f(K) = 1110 \oplus 1001 = 0111. \text{ Совпадет с исходным текстом } P$$

Функция f (101) = 1001 является неконвертируемой, но операция ИСКЛЮЧАЮЩЕЕ ИЛИ позволяет нам использовать функцию и в алгоритмах дешифрования, и в шифровании. Другими словами, функция является неконвертируемой, но смеситель будет самоконвертируемым.

Усовершенствование. Попробуем улучшить нашу первую идею, чтобы приблизиться к шифру Файстеля. Мы знаем, что должны применить вход к неконвертируемому элементу (функции), но мы не будем использовать только ключ. Мы задействуем также вход к функции, чтобы применить ее для шифрования части исходного текста и дешифрования части зашифрованного текста. Ключ может использоваться как второй вход к функции. Этим способом наша функция становится сложным элементом с некоторыми неключевыми

элементами и некоторыми ключевыми элементами. Чтобы достичь цели, разделим исходный текст и зашифрованный текст на два блока равной длины – левый (L) и правый (R). Правый блок вводится в функцию, а левый блок складывается с помощью операции ИСКЛЮЧАЮЩЕЕ ИЛИ с выходом функции. Мы должны запомнить, что входы к функции должны точно совпадать в шифровании и дешифровании. Это означает, что правая секция исходного текста до шифрования и правая секция зашифрованного текста после дешифрования будут совпадать. Другими словами, секция должна войти в шифрование и выйти из дешифрования неизмененной. Рисунок 7.16 иллюстрирует идею.

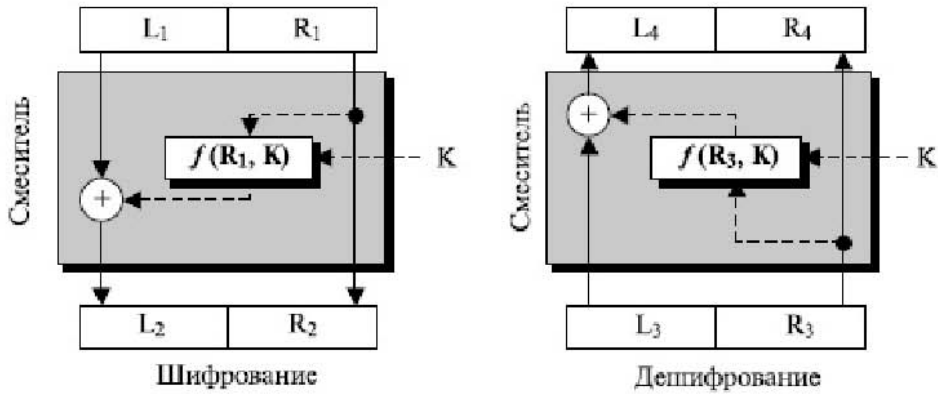


Рис. 7.16. Усовершенствование предыдущей схемы Файстеля

Алгоритмы шифрование и дешифрования инверсны друг другу. Предположим, что $L_3 = L_2$ и $R_3 = R_2$ (в зашифрованном тексте в течение передачи не произошло изменений).

$$R_4 = R_3 = R_2 = R_1$$
$$L_4 = L_3 \oplus f(R_3, K) = L_2 \oplus f(R_2, K) = L_1 \oplus f(R_1, K) \oplus f(R_1, K) = L_1$$

Исходный текст, используемый в алгоритме шифрования, — это текст, правильно восстановленный алгоритмом дешифрования.

Окончательный вариант. Предыдущее усовершенствование имеет один недостаток: правая половина исходного текста никогда не изменяется. Ева может немедленно найти правую половину исходного текста, разбивая на части зашифрованный текст и распаковывая его правую

половину. Проект нуждается в дальнейших шагах усовершенствования.

Первое: увеличим *число раундов*. Второе: добавим новый элемент в каждый раунд — устройство замены. Эффект устройства замены в раунде шифрования компенсируется эффектом устройства замены в раунде дешифрования. Однако это позволяет нам менять левые и правые половины в каждом раунде. Рисунок 7.17 иллюстрирует новый вариант шифра Файстеля с двумя раундами.

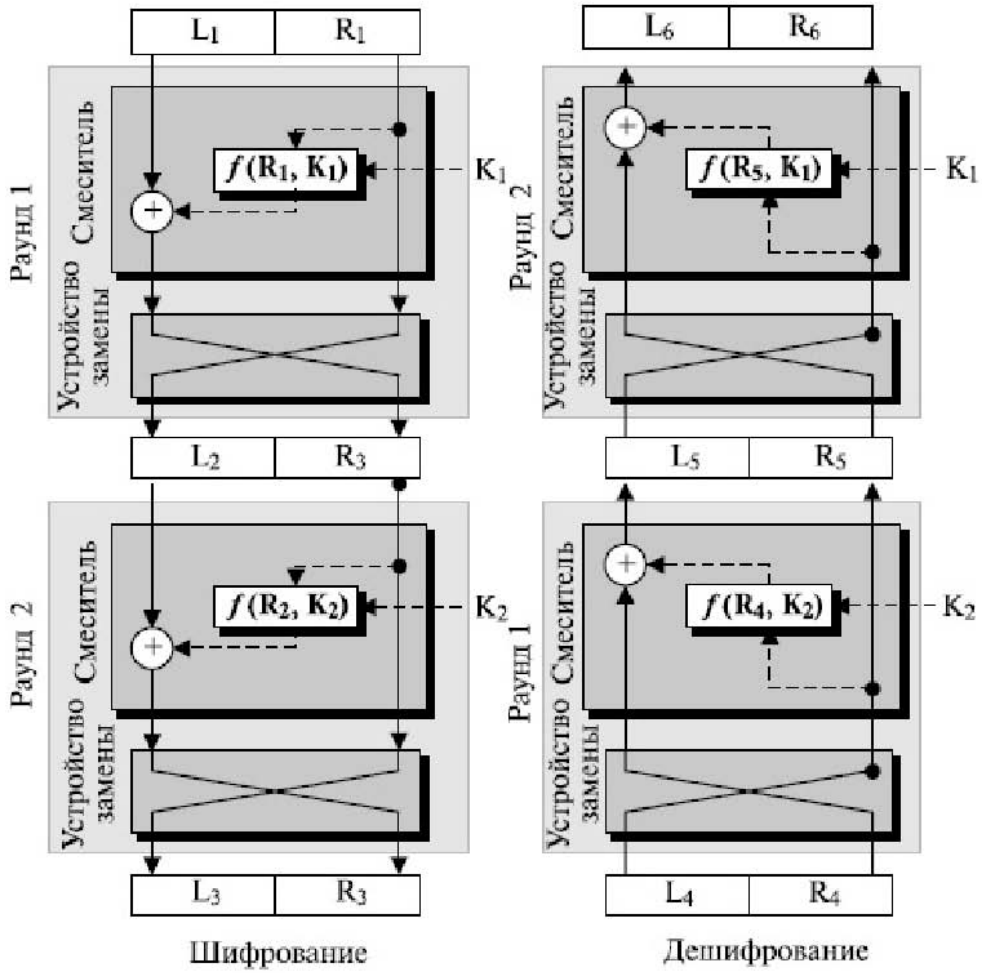


Рис. 7.17. Окончательный вариант шифра Файстеля с двумя раундами

Обратите внимание, что есть два ключа раундов: K_1 и K_2. Ключи используются в обратном порядке в шифровании и дешифровании.

Поскольку два смесителя инверсны друг другу и устройства замены инверсны друг другу, очевидно, что шифрование и дешифрование также инверсны друг другу. Однако мы можем доказать этот факт, используя отношения между левыми и правыми секциями в каждом шифре. Другими словами, если $L_6 = L_1$ и $R_6 = R_1$, предположим, что $L_4 = L_3$ и $R_4 = R_3$ (шифрованный текст не изменился при передаче). Вначале докажем это для промежуточного текста:

$$L_5 = R_4 \oplus f(L_4, K_2) = R_3 \oplus f(R_2, K_2) = L_2 \oplus f(R_2, K_2) \oplus f(R_2, K_2) = L_2$$
$$R_5 = L_4 = L_3 = R_2$$

Тогда просто доказать равенство для двух блоков исходного текста.

$$L_6 = R_5 \oplus f(L_5, K_1) = R_2 \oplus f(L_2, K_1) = L_1 \oplus f(R_1, K_1) \oplus f(R_1, K_1) = L_1$$
$$R_6 = L_5 = L_2 = R_1$$

Шифры не-Файстеля

Шифр не-Файстеля использует только обратимые компоненты. Компонент в исходном тексте имеет соответствующий компонент в шифре. Например, S -блоки должны иметь равное число входов и выходов, чтобы быть совместимыми. Не позволяется никакое сжатие или расширение P -блоков, потому что они станут необратимыми. В шифре не-Файстеля нет потребности делить исходный текст на две половины, как мы видели в шифрах Файстеля.

Рисунок 7.13 можно рассматривать как графическую иллюстрацию принципа шифра не-Файстеля, потому что единственные компоненты в каждом раунде — самообратимые операции ИСКЛЮЧАЮЩЕЕ ИЛИ, S -блоки 2×2, которые могут спроектированы, чтобы быть обратимыми, и прямые P -блоки, которые обратимы, если использована соответствующая таблица перестановки. Поскольку каждый компонент является обратимым, то можно показать, что и каждый раунд является обратимым. Мы только должны применять ключи раундов в обратном порядке. Шифрование использует ключи раундов K_1 и K_2. Алгоритм дешифрования должен пользоваться ключами раундов K_2 и K_1.

Атаки на блочные шифры

Атаки традиционных шифров могут также использоваться для современных *блочных шифров*, но сегодняшние *блочные шифры* успешно противостоят большинству атак, обсужденных в лекциях 5-6. Например, грубая силовая атака ключа, как правило, неосуществима, потому что ключи обычно имеют очень большую длину. Однако недавно были изобретены некоторые новые виды атак *блочных шифров*, которые основаны на структуре современных *блочных шифров*. Эти атаки используют методы и дифференциального, и линейного анализа.

Дифференциальный криптоанализ

Идею относительно дифференциального криптоанализа предложили Эли Бихам и Ади Шамир. Это — атака с выборкой исходного текста. Ева может каким-либо образом получить доступ к компьютеру Алисы и завладеть выборочно частью исходного текста и соответствующего зашифрованного текста. Цель состоит в том, чтобы найти ключ шифра Алисы.

Алгоритм анализа. Перед тем как Ева предпримет атаку с выборкой исходного текста, она должна проанализировать алгоритм шифрования, чтобы собрать некоторую информацию об отношениях зашифрованного и исходного текстов. Очевидно, Ева не знает ключ шифра. Однако некоторые шифры имеют слабости в структурах, которые могут позволить Еве найти различия исходного текста и различия зашифрованного текста, не зная ключ.

Предположим, что шифр состоит только из одной операции ИСКЛЮЧАЮЩЕЕ ИЛИ, как показано на рис. 7.18. Не зная значения ключа, Ева может легко найти отношения между разностями исходного текста и разностями зашифрованного текста. Если разность исходного текста мы обозначим $P_1 \oplus P_2$ и разность зашифрованного текста мы обозначим $C_1 \oplus C_2$, приведенные следующие преобразования доказывают, что $C_1 \oplus C_2 = P_1 \oplus P_2$:

$$C_1 = P_1 \oplus K C_2 = P_2 \oplus K \rightarrow C_1 \oplus C_2 = P_1 \oplus K \oplus P_2 \oplus K = P_1 \oplus P_2$$

Однако этот пример нереалистичен; модемные *блочные шифры* не настолько просты.

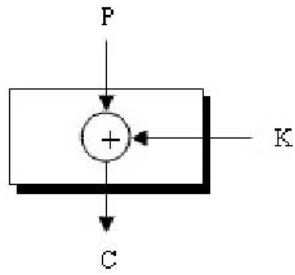
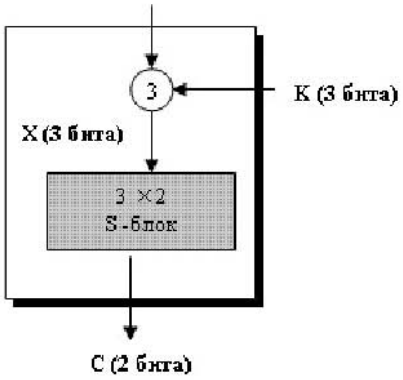


Рис. 7.18. Диаграмма для примера 7.13

В примере 7.13 мы добавляем один S -блок, как показано на рис. 7.19.



X	000	001	010	011	100	101	110	111
C	11	00	10	10	01	00	11	00

Таблица S - блока

Рис. 7.19. Диаграмма для примера 7.14

Хотя эффект шифрования ключом не действует, когда мы используем разности между двумя X и двумя P $(X_1 \oplus X_2 = P_1 \oplus P_2)$, существование S -блока мешает Еве найти и определенные отношения между разностями исходного текста и разностями зашифрованного текста. Однако возможно установить вероятностные отношения. Ева может составить таблицу 7.4, которая показывает для разности исходного текста, сколько можно создать разностей зашифрованного текста — шифр. Обращаем внимание, что таблица сделана по информации, которая произведенеа с учетом таблицы входа-выхода S -

блока по рис. 7.19, потому что $P_1 \oplus P_2 = X_1 \oplus X_2$.

Таблица 7.4. Дифференциальная таблица для входов и выходов для шифра в примере 7.14

		$C_1 \oplus C_2$			
		00	01	10	11
$P_1 \oplus P_2$	000	8			
	001	2	2		4
	010	2	2	4	
	011		4	2	2
	100	2	2	4	
	101		4	2	2
	110	4		2	2
	111			2	6

Поскольку размер ключей — 3 бита, может быть восемь случаев для каждой разности во вводе. Таблица показывает, что если входная разность — $(000)_2$, разность выхода — всегда $(00)_2$. С другой стороны, таблица показывает, что если входная разность — $(100)_2$, то имеется два случая разностей выхода $(00)_2$, два случая разностей выхода $(01)_2$ и четыре случая разностей выхода $(01)_2$.

Пример 7.15

Эвристический результат примера 7.14 может создать вероятностную информацию для Евы, как показано в таблице 7.5. Входы в таблице соответствуют вероятностям появления. Разности с нулевой вероятностью никогда не будут возникать.

Таблица 7.5. Дифференциальная таблица для входов и

выходов для шифра в примере 7.15

	00	01	10	11
000	1	0	0	0
001	0,25	0,25	0	0,50
010	0,25	0,25	0,50	0
011	0	0,50	0,25	0,25
100	0,25	0,25	0,50	0
101	0	0,50	0,25	0,25
110	0,50	0	0,25	0,25
111	0	0	0,25	0,75

Как мы увидим позже, Ева теперь располагает достаточным количеством информации, чтобы начать ее атаку. Таблица показывает, что вероятности распределены неоднородно из-за слабости в структуре `S` -блока. Таблица 7.5 упоминается иногда как дифференциальная таблица распределения или профайл ИСКЛЮЧАЮЩЕЕ ИЛИ.

Запуск атаки выборки исходного текста. После того как анализ однажды сделан, он может быть сохранен для будущего использования, пока структура шифра не изменится. Ева может выбрать для атак исходные тексты. Дифференциальная таблица *распределения вероятности* (таблица 7.5) поможет Еве их выбирать — она возьмет те, которые имеют самую высокую вероятность в таблице.

Предположительное значение ключа. После запуска некоторых атак с соответствующей выборкой исходного текста Ева может найти некоторую пару "исходный текст / зашифрованный текст", которая позволяет ей предположить некоторое значение ключа. Процесс начинается от `C` и продвигается к `P`.

Пример 7.16

Рассматривая таблицу 7.5, Ева знает, что если $P_1 \oplus P_2 = 001$, то $C_1 \oplus C_2 = 11$ с вероятностью `0,50` (`50` процентов). Она пробует взять $C_1 = 00$ и получает $P_1 = 010$ (атака с выборкой

зашифрованного текста). Она еще пробует $C_2 = 11$ и получает $P_2 = 011$ (другая атака с выборкой зашифрованного текста). Теперь она пробует вернуться к анализу, основанному на первой паре, P_1 и C_1:

$$C_1 = 00 \rightarrow X_1 = 001 \text{ или } X_1 = 111$$
$$\text{Если } X_1 = 001 \rightarrow K = X_1 \oplus P_1 = 011. \rightarrow \text{Если } X_1 = 111 \rightarrow K = X_1 \oplus P_1 = 101$$

Используя пару P_2 и C_2, получим

$$C_2 = 11 \rightarrow X_2 = 000 \text{ или } X_1 = 110$$
$$\text{Если } X_2 = 000 \rightarrow K = X2 \oplus P2 = 011 \rightarrow \text{Если } X_{12} = 110 \rightarrow K = X2 \oplus P2 = 101$$

Два испытания показывают, что $K = 011$ или $K = 101$. Хотя Ева не уверена, какое из них точное значение ключа, она знает, что самый правый бит — 1 (общий бит между двумя значениями). Продолжая атаку, можно учитывать, что самый правый бит в ключе – 1. Таким образом можно определить другие биты в этом ключе.

Общая процедура. Современные *блочные шифры* имеют большую сложность, чем, та, которую мы обсуждали в этом разделе. Кроме того, они могут содержать различное количество раундов. Ева может использовать следующую стратегию:

1. Поскольку каждый раунд содержит одни и те же операции, Ева может создать таблицу дифференциальных распределений (профайл ИСКЛЮЧАЮЩЕГО ИЛИ) для каждого S -блока и комбинировать их, чтобы создать распределение для каждого раунда.
2. Предположим, что каждый раунд независим (справедливое предположение). Ева может создать таблицу распределения для всего шифра, умножая соответствующие вероятности.
3. Ева может теперь делать список исходных текстов для атак, основанных на таблице распределений на втором шаге. Заметим, что таблица в шаге 2 только помогает Еве выбирать меньшее количество пар "исходный текст / зашифрованный текст"
4. Ева выбирает зашифрованный текст и находит соответствующий

исходный текст. Затем она анализирует результат, чтобы найти некоторые биты в ключе.

5. Ева повторяет шаг 4, чтобы найти больше битов в ключе.
6. После нахождения достаточного количества битов в ключе Ева может использовать атаку грубой силы, чтобы найти весь ключ.

Дифференциальный криптоанализ базируется на таблице неоднородных дифференциальных распределений, S -блоков в блочном шифре.
Более детально дифференциальный криптоанализ приводится в Приложении N.

Линейный криптоанализ

Линейный криптоанализ был представлен Митцури Мацуи (Mitsuru Matsui) в 1993 году. Анализ использует атаки знания исходного текста (в отличии от атак с выборкой исходного текста в дифференциальном криптоанализе). Полное обсуждение этой атаки базируется на некоторых понятиях теории вероятностей, которые находятся за рамками этой книги. Чтобы рассмотреть главную идею этой атаки, предположим, что шифр состоит из одного раунда, как показано на рис. 7.20, где c_0, c_1 и c_2 представляют три бита на выходе и x_0, x_1 и x_2 представляют три бита на входе S -блока.

S -блок — линейное преобразование, в котором каждый вывод является линейной функцией ввода, как мы обсуждали ранее в этой лекции. С этим линейным компонентом мы можем создать три *линейных уравнения* между исходным текстом и битами зашифрованного текста, как показано ниже:

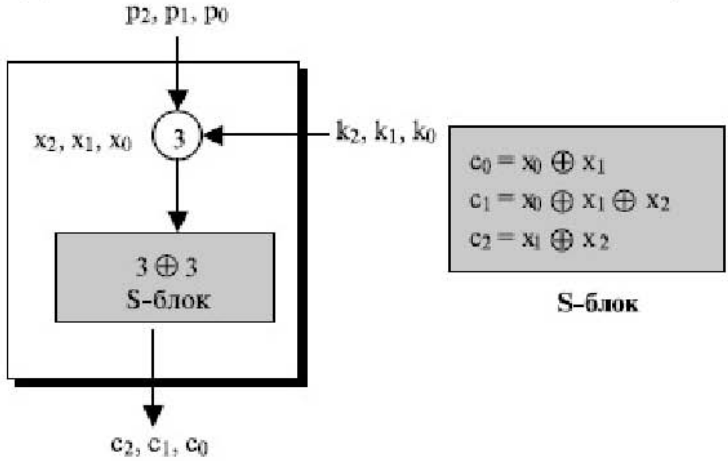


Рис. 7.20. Простой шифр с линейным S-блоком

$$c_0 = p_0 \oplus k_0 \oplus p_1 \oplus k_1$$
$$c_1 = p_0 \oplus k_0 \oplus p_1 \oplus k_1 \oplus p_2 \oplus k_2$$
$$c_2 = p_1 \oplus k_1 \oplus p_2 \oplus k_2 \oplus$$

Решая систему уравнений для трех неизвестных, мы получаем

$$k_1 = (p_1) \oplus (c_0 \oplus c_1 \oplus c_2)$$
$$k_2 = (p_2) \oplus (c_0 \oplus c_1)$$
$$k_0 = (p_0) \oplus (c_1 \oplus c_2)$$

Это означает, что три атаки типа "знания исходного текста" могут найти значения k_1, и k_2. Однако реальные *блочные шифры* не так просты, как этот; они имеют больше компонентов, и S -блоки не линейны.

Линейная аппроксимация. В некоторых современных *блочных шифрах* может случиться, что некоторые S -блоки не полностью нелинейные; тогда они могут быть в вероятностном смысле аппроксимированы некоторыми *линейными функциями*. Вообще, задавая исходный и зашифрованный текст в n бит и ключ m бит, мы ищем некоторые уравнения, имеющие вид

$$(k_0 \oplus k_1 \oplus \cdots \oplus k_x) = (p_0 \oplus p_1 \oplus \cdots \oplus p_y) \oplus (c_0 \oplus c_1 \oplus \cdots \oplus c_z)$$

7.2. Современные шифры потока

В лекциях 5-6 мы кратко обсуждали разницу между традиционными шифрами потока и традиционными *блочными шифрами*. Подобные отличия существуют также между современными шифрами потока и современными *блочными шифрами*. В современном шифре потока шифрование и дешифрование проводятся r бит одновременно. Мы имеем поток бит исходного текста $P = p_n \ldots p_2 p_1$, поток бит зашифрованного текста $C = c_n \ldots c_2 c_1$, и ключевой поток бит $K = k_n \ldots k_2 k_1$, в которых p_i c_i, и k_i — это r -битные слова. Шифрование — $c_i = E(k_i, p_i)$, и дешифрование — $p_i = D(k_i, c_i)$, как показывает рис. 7.21.

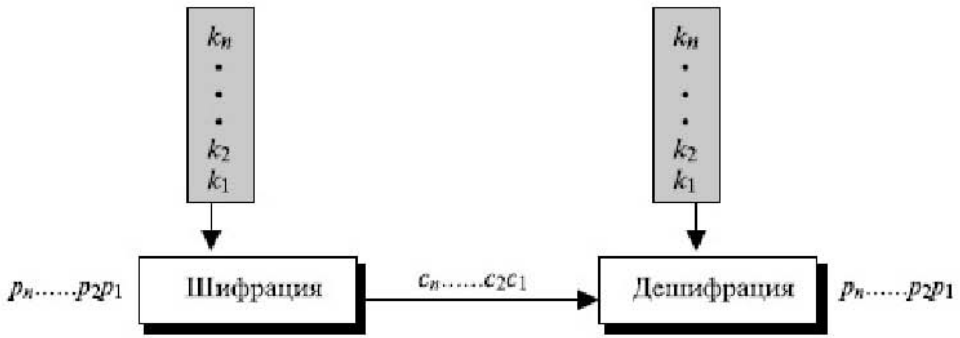


Рис. 7.21. Шифр потока

Шифры потока быстрее, чем *блочные шифры*. Аппаратная реализация шифра потока также более проста. Когда мы должны зашифровать двоичные потоки и передать их на постоянной скорости, лучший выбор — использовать шифр потока. Шифры потока обладают большей защитой против искажения битов в течение передачи.

В современном шифре потока каждое r -битовое слово в потоке исходного текста зашифровано, используя r -битовое слово в ключевом потоке, чтобы создать соответствующее r -битовое слово в потоке зашифрованного текста.

Рассматривая рис. 7.21, можно предположить, что главная проблема в современных шифрах потока — как генерировать ключевой поток $K =$

$k_n \ldots k_2 k_1$. Современный шифр потока можно разделить на две обширные категории: синхронный и несинхронный.

Синхронные шифры потока

В синхронном шифре потока ключевой поток независим от потока зашифрованного текста или исходного текста. Ключевой генерируемый поток не используется без отношений между ключевыми битами и исходным текстом или битами зашифрованного текста.

В синхронном шифре потока ключ независим от исходного текста или зашифрованного текста.

Одноразовый блокнот

Наиболее простой и самый безопасный тип синхронного шифра потока назван шифром одноразового блокнота, или, по имени изобретателя, "шифром Вернама". Шифр одноразового блокнота использует ключевой поток, который беспорядочно выбран для каждой шифровки. Алгоритмы шифрования и дешифрования применяют единственную операцию — ИСКЛЮЧАЮЩЕЕ ИЛИ. Шифры, которые базируются на свойствах операции ИСКЛЮЧАЮЩЕЕ ИЛИ, обсуждались ранее. В них алгоритмы шифрования и дешифрования инверсны друг другу. Важно, что в этом шифре операция ИСКЛЮЧАЮЩЕЕ ИЛИ используется только для одного бита одновременно. Другими словами, операция производится над словом не более чем из одного бита и полем $GF(2)$. Заметим, что также нужно иметь безопасный канал для того, чтобы Алиса могла передать ключевую последовательность потока Бобу (рис. 7.22).

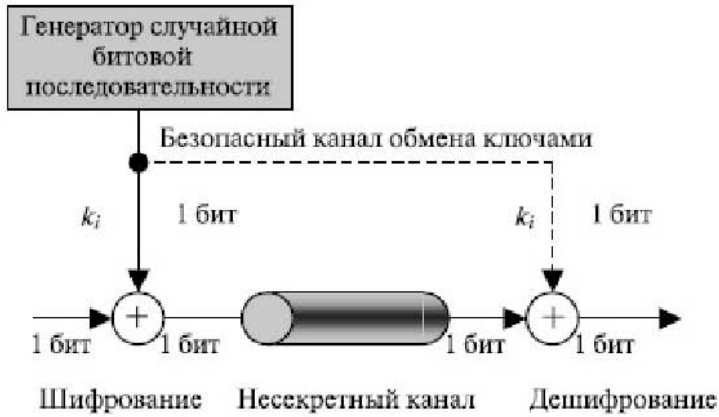


Рис. 7.22. Одноразовый блокнот

Одноразовый блокнот — идеальный шифр. Он совершенен. Нет метода, который дал бы противнику возможность распознать ключ или статистику зашифрованного текста и исходного текста. Нет никаких соотношений между исходным и зашифрованным текстами. Другими словами, зашифрованный текст – настоящий случайный поток битов, даже если получить некоторые образцы исходного текста. Ева не может нарушить шифр, если она не попробует все возможные случайные ключевые потоки, которые были бы 2^n, если размер исходного текста — n -битовый. Однако при этом есть проблема. Передатчик и приемник для того, чтобы совместно использовать одноразовый блокнот ключей, должны установить соединение каждый раз, когда они хотят обменяться информацией. Они должны, так или иначе, договориться о случайном ключе. Так что этот совершенный и идеальный шифр очень трудно реализовать.

Пример 7.17

Какой вид имеет зашифрованный текст при использовании шифра одноразового блокнота в каждом из следующих случаев?

а. Исходный текст состоит из n нулей.

б. Исходный текст состоит из n единиц.

в. Исходный текст состоит из чередующихся нулей и единиц.

г. Исходный текст — случайная строчка бит.

Решение

a. Поскольку $0 \oplus k_i = k_i$, то поток зашифрованного текста совпадет с ключевым потоком. Если ключ случайный, зашифрованный текст также случайный. Отрывки исходного текста в зашифрованном тексте не сохраняются.

b. Поскольку $1 \oplus k_i = \bar{k}_i$, где $\bar{k}_i$ является дополнением, поток зашифрованного текста — дополнение ключевого потока. Если ключевой поток случайный, то зашифрованный текст также случайный, отрывки исходного текста не сохраняются в зашифрованном тексте.

c. В этом случае каждый бит в потоке зашифрованного текста является или тем же самым, что и в ключевом потоке, или его дополнением. Поэтому результат — также случайный, если ключевой поток случайный.

d. В данном случае зашифрованный текст явно случайный, потому что проведение операции ИСКЛЮЧАЮЩЕЕ ИЛИ двух случайных битов в результате дает случайный поток бит.

Регистр сдвига с обратной связью

Одно усовершенствование к одноразовому блокноту — Регистр сдвига с обратной связью (*FSR — Feedback Shift Register*). *FSR* может быть реализован или в программном обеспечении, или в аппаратных средствах, но для простоты мы рассмотрим аппаратную реализацию. Регистр сдвига с обратной связью состоит из регистра сдвига и функции обратной связи, как показано на рис. 7.23.

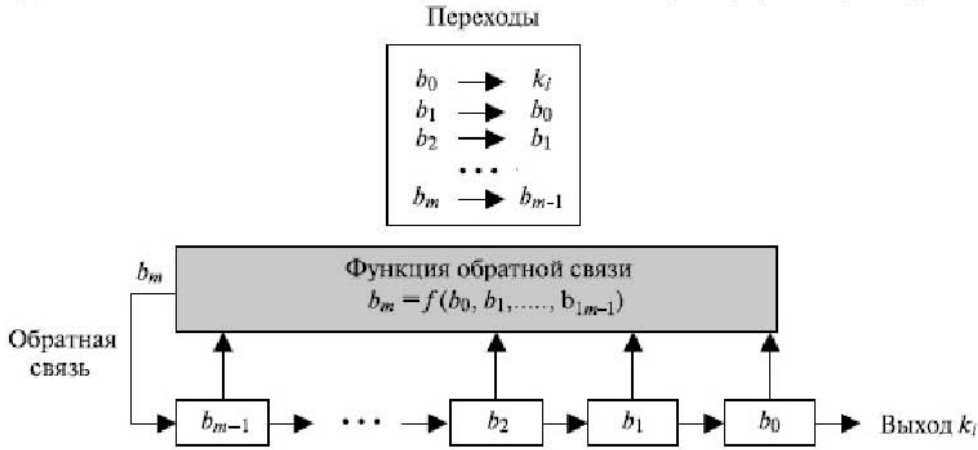


Рис. 7.23. Регистр сдвига с обратной связью (FSR)

Регистр сдвига – последовательность из m ячеек от b_0 до b_{m-1}, где каждая ячейка предназначена для сохранения единственного бита. Ячейки рассматриваются как n -битовое слово, называемое в начале "начальное значение" или источник. Всякий раз, когда необходимо получить бит на выходе (например, по сигналу в определенное время), каждый бит сдвигается на одну ячейку вправо. Это означает, что значение каждой ячейки присваивается правой соседней ячейке и принимает значение левой ячейки. Самая правая ячейка b_0 считается выходом и дает выходное значение (k_i). Крайняя левая ячейка, b_{m-1}, получает свое значение согласно значению информации функции обратной связи. Обозначаем выход функции с информацией обратной связи b_m. Функция информации обратной связи определяет, какие значения имеют ячейки, чтобы вычислить b_m. *Регистр сдвига* информации обратной связи может быть линейный или нелинейный.

Линейный регистр сдвига с обратной связью (LFSR). Примем, что b_m — это линейная функция b_0, b_1, ……, b_{m-1}, для которой

$$b_m = c_{m-1}b_{m-1} + \cdots + c_2 b_2 + c_1 b_1 + c_0 b_0 \ (c_0 \neq 0)$$

Линейный *регистр сдвига* с обратной связью работает с двоичными цифрами, поэтому умножение и сложение находятся в поле $GF(2)$, так что значение C_i является или 1, или 0, но C_0 должно быть 1, чтобы

получить информацию обратной связи на выходе. Операция сложения – это операция ИСКЛЮЧАЮЩЕЕ ИЛИ. Другими словами,

$$b_m = c_{m-1}b_{m-1} \oplus \cdots \oplus c_2 b_2 \oplus c_1 b_1 \oplus c_0 b_0 \ (c_0 \neq 0)$$

Пример 7.18

Построим линейный *регистр сдвига* с обратной связью с 5 -ю ячейками, в которых $b_5 = b_4 \oplus b_2 \oplus b_0$.

Решение

Если $c_i = 0$, b_i не играет роли в вычислении b_m, то это означает, что b_i не связан с функцией информации обратной связи. Если $c_i = 1$, b_i включается в вычисление b_m. В этом примере c_1 и c_3 — нули, это означает, что, мы имеем только три подключения. Рисунок 7.24 показывает схему линейного *регистра сдвига* с обратной связью.

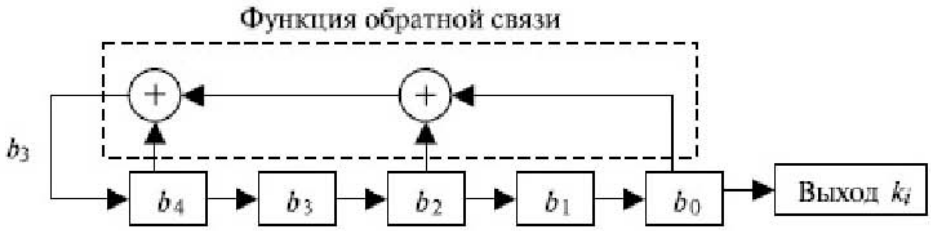


Рис. 7.24. Линейный регистр сдвига с обратной связью

Пример 7.19

Построим линейный *регистр сдвига* с обратной связью с 4 -мя ячейками, в которых $b_4 = b_1 \oplus b_0$. Покажите значение регистра после 20 *операций (сдвигов)*, если исходное значение — $(0001)_2$.

Решение

Рисунок 7.25 показывает схему и использование линейного *регистра сдвига* с обратной связью для шифрования.

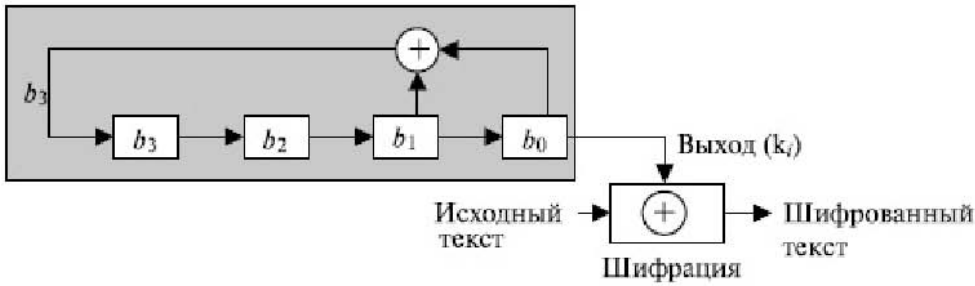


Рис. 7.25. Линейный регистр сдвига с обратной связью для примера 7.19

Таблица 7.6. показывает значение потока ключей. Для каждого перехода первое значение b_4 вычисляется, а затем каждый бит сдвигается на одну ячейку вправо.

Таблица 7.6.

Текущее значение	b_4	b_3	b_2	b_1	b_0	k_i
Начальное значение	1	0	0	0	1	
1	0	1	0	0	0	1
2	0	0	1	0	0	0
3	1	0	0	1	0	0
4	1	1	0	0	1	0
5	0	1	1	0	0	1
6	1	0	1	1	0	0
7	0	1	0	1	1	0
8	1	0	1	0	1	1
9	1	1	0	1	0	1
10	1	1	1	0	1	0
11	1	1	1	1	0	1
12	0	1	1	1	1	0
13	0	0	1	1	1	1
14	0	0	0	1	1	1
15	1	0	0	0	1	1
16	0	1	0	0	0	1

17	0	0	1	0	0	0
18	1	0	0	1	0	0
19	1	1	0	0	1	0
20	1	1	1	0	0	1

Заметим, что поток ключей — `1000100110101111 1001……` . Он выглядит, на первый взгляд, как случайная последовательность, но если просмотреть большое число транзакций (сдвигов), мы можем увидеть, что последовательности периодичны. Это повторение по `15` бит показано ниже.

1000100110101111 000100110101111 0001001101011111 **000100110101111**

Ключ потока генерирует с помощью линейного *регистра сдвига* с обратной связью *псевдослучайную последовательность*, в которой повторяются последовательности длиною `N`. Поток периодичен. Он зависит от схемы генератора и начальной информации и может быть не свыше $2^m - 1$. Каждая схема порождает `m` -битовые последовательности от содержащих все нули до содержащих все единицы. Однако если начальная последовательность состоит только из нулей, результат бесполезен — исходный текст был бы потоком из одних нулей. Поэтому такая начальная последовательность исключена.

Максимальный период последовательностей, которые генерируются с помощью линейного регистра сдвига с обратной связью, — $2^m - 1$.

В предыдущем примере максимальный период — ($2^4 - 1 = 15$). Чтобы достичь этой максимальной периодичности (наилучшей рандомизации), мы должны в первую очередь представить функцию обратной связи как характеристический полином с коэффициентами в поле `GF(2)`.

$$b_m = c_{m-1}b_{m-1} + \ldots + c_1b_1 + c_0b_0 \rightarrow x^m = c_{m-1}x^{m-1} + \ldots + c_1x^1 + c_0x^0$$

Поскольку сложение и вычитание в этом поле одни и те же, все элементы могут быть перенесены в одну сторону, что дает *полином* степени `m`. (называемый характеристическим полиномом).

$$x^m + c_{m-1}x^{m-1} + \ldots + c_1x^1 + c_0x^0 = 0$$

Линейный *регистр сдвига* с обратной связью имеет максимальный период 2^m-1, если он имеет четное число ячеек, и характеристический *полином* — примитивный полином. Примитивный *полином* — неприводимый *полином*, который является делителем $x^e - 1$, где e — наименьшее целое число в форме $e = 2^k-1$ и $k >= 2$. Примитивный *полином* получить нелегко. *Полином* выбирается случайно, а затем проверяется на примитивность. Однако существуют таблицы проверенных примитивных полиномов (см. приложение G).

Пример 7.20

Характеристический *полином* для линейного *регистра сдвига* с обратной связью в примере 7.19 — ($x^4 + x + 1$) — является примитивным полиномом. Таблица 5.1 (лекция 5) показывает, что это — неприводимый *полином*. Этот *полином* также делит $(x^7 + 1) = (x^4 + x + 1)\ (x^3 + 1)$, что означает $e = 2^3 - 1 = 7$.

Атаки шифров, полученных с помощью линейных регистров сдвига с обратной связью.Линейный регистр сдвига с обратной связью имеет очень простую структуру, но эта простота делает шифр уязвимым к атакам. Два общих типа атаки приведены ниже.

1. Если структура линейного *регистра сдвига* с обратной связью известна, то после перехвата и анализа одного n -битового куска зашифрованного текста Ева может предсказать все будущие зашифрованные тексты.
2. Если структура линейного *регистра сдвига* с обратной связью неизвестна, Ева может использовать атаку знания исходного текста длиной $2n$ бит, чтобы вскрыть шифр.

Нелинейный регистр сдвига с обратной связью. Линейный *регистр сдвига* с обратной связью уязвим главным образом из-за его линейности. Более устойчивый шифр потока может быть получен при использовании нелинейного регистра сдвига с обратной связью(NLFSR). Он имеет ту же самую структуру, что и линейный *регистр сдвига* с обратной связью, за исключением того, что b_m —

нелинейная функция $b_0, b_1, \ldots, b_m$. Например, в 4 -битном нелинейном регистре сдвига с обратной связью структура определяется соотношением, показанным ниже, где операция AND означает поразрядную операцию И, а OR означает поразрядную операцию ИЛИ. Черточка над переменной означает инверсию.

$$b_4 = (b_3 \text{ AND } b_2) \text{ or } (b_1 \text{ AND } \overline{b_0})$$

Однако нелинейный *регистр сдвига* с обратной связью не имеет общего характера, поскольку нет математического обоснования, как получить такой регистр с максимальным периодом.

Можно применить линейный *регистр сдвига* с обратной связью с максимальным периодом и затем скомбинировать его обратную связь с помощью нелинейной функции.

Несинхронные шифры потока

В несинхронном шифре потока каждый ключ в ключевом потоке зависит от предыдущего исходного текста или зашифрованного текста.

В несинхронном шифре потока ключ зависит либо от исходного текста, либо от зашифрованного текста.

Два метода, которые используются, чтобы создать различные режимы работы для *блочных шифров* (режим обратной связи по выходу и режим счета сцеплений блоков шифра), фактически создают шифры потока (см. лекцию 11).

7.3. Рекомендованная литература

Нижеследующие книги и сайты дают более детальные сведения по обсуждаемым вопросам, которые рассмотрены в этой лекции. Ссылки, помещенные в скобки, приведены в списке в конце книги.

Книги

[Sti06] и [PHS03] содержат полные сведения о P -блоках и S -блоках. *Поточные шифры* тщательно рассмотрены в [Sch99] и [Sal03]. [Sti06], [PHS03] и [Vau06] — полный и интересный анализ дифференциального и *линейного криптоанализа*.

Сайты

Нижеследующие сайты дают более подробную информацию о темах, обсужденных в этой лекции.

- ссылка: http://en.wikipedia.org/wiki/FeisteL.cipher - http://en.wikipedia.org/wiki/FeisteL.cipher
- ссылка: http://www.quadibloc.com/crypto/co040906.htm - http://www.quadibloc.com/crypto/co040906.htm
- ссылка: tigger.uic.edu/~jleon/mcs425-s05/handouts/feistal-diagram.pdf - tigger.uic.edu/~jleon/mcs425-s05/handouts/feistal-diagram.pdf

7.4. Итоги

- Традиционные шифры с *симметричным ключом* — шифры, ориентированные на символ. С появлением компьютера стали нужны шифры, ориентированные на биты.
- Современный симметричный ключевой *блочный шифр* зашифровывает n -битный блок исходного текста или расшифровывает n -битовый блок зашифрованного текста. Алгоритмы шифрования или дешифрования используют k -битные ключи.
- Современный *блочный шифр* может быть спроектирован так, чтобы действовать как шифр подстановки или шифр транспозиции. Однако чтобы быть стойким к атаке исчерпывающего поиска, современный *блочный шифр* должен быть спроектирован как шифр подстановки.
- Современные *блочные шифры* — обычно ключевые шифры подстановки, в которых ключ практически позволяет отображение всех возможных входов во все возможные выходы.
- Современный *блочный шифр* состоит из комбинации P -блоков, модулей подстановки, S -блоков и некоторых других модулей.

- P -блок (блок перестановки) подобен традиционному шифру транспозиции для символов. Есть три типа P -блоков: прямые P -блоки, P -блоки расширения и P -блоки сжатия.
- S -блок (блок подстановки) можно представить себе как маленький блок шифра подстановки. Однако в S -блоке может быть различное число входов и выходов.
- Операция ИСКЛЮЧАЮЩЕЕ ИЛИ — важный компонент в большинстве *блочных шифров*: она представляет операции сложения или вычитания в поле GF (2).
- В современных *блочных шифрах* часто применяется *операция циклического сдвига*, в которой смещение может быть влево или вправо. Операция перестановки — специальный случай *операции циклического сдвига*, где k = n/2. Две других операции, применяемые в некоторых *блочных шифрах*, — разбиение и комбинирование.
- Шеннон ввел понятие составного шифра. Составной шифр — сложный шифр, объединяющий S -блоки, P -блоки и другие компоненты, чтобы достигнуть рассеивания и перемешивания. Рассеивание скрывает отношения между исходным текстом и зашифрованным текстом, перемешивание скрывает отношения между ключом шифра и зашифрованным текстом.
- Современные *блочные шифры* — все составные шифры, но они разделены на два класса: шифры не-Файстля и шифры Файстеля. Шифры Файстеля используют и обратимые, и *необратимые* компоненты. Шифры не-Файстля используют только обратимые компоненты.
- Некоторые новые атаки *блочных шифров* базируются на структуре современных шифров. Эти атаки используют дифференциальные и линейные методы *криптоанализа*
- В современном шифре потока каждое слово r -бита в потоке исходного текста зашифровано, для чего используется r -битовое слово в потоке ключей, чтобы создать соответствующее r -битовое слово в потоке зашифрованного текста. Современные шифры потока могут быть разделены на две обширные категории: синхронные шифры потока и несинхронный шифр потока в синхронном шифре потока. В первом случае ключевой поток независим от потока зашифрованного текста или исходного текста. В несинхронном шифре потока ключевой поток зависит от

исходного текста или потока зашифрованного текста.

- Самый простой и самый безопасный тип синхронного шифра потока назван одноразовым блокнотом. Шифр одноразового блокнота использует ключевой поток ключей, который выбран беспорядочно для каждого шифрования. Алгоритмы шифрования и дешифрования используют операцию ИСКЛЮЧАЮЩЕЕ ИЛИ. Шифр одноразового блокнота не годится для практики, потому что ключ должен быть индивидуальным для каждого сеанса связи. Один из компромиссных вариантов одноразового блокнота — *регистр сдвига* с обратной связью (*FSR*), который может быть реализован в аппаратных средствах или программном обеспечении.

7.5. Вопросы и упражнения

Обзорные вопросы

1. Укажите различия между современным и традиционным шифрами с *симметричным ключом*.
2. Объясните, почему современные *блочные шифры* спроектированы как шифры подстановки вместо того, чтобы применять шифры транспозиции.
3. Объясните, почему шифр подстановки можно представить себе как шифр транспозиции.
4. Перечислите некоторые компоненты современного *блочного шифра*.
5. Определите `P` -блок и перечислите его три варианта. Какой вариант является обратимым?
6. Определите `S` -блок и покажите необходимое условие обратимости `S` -блока.
7. Определите составной шифр и перечислите два класса составных шифров.
8. Укажите различие между рассеиванием и перемешиванием
9. Укажите различие между блочным шифром Файстеля и не-Файстеля.
10. Укажите различие между дифференциальным и *линейным криптоанализом*. Какой из них использует атаку выборки

исходного текста? Какой из них использует также атаку знания исходного текста?

11. Укажите различие между синхронным и несинхронным шифрами потока.
12. Определите *регистр сдвига* с обратной связью и перечислите два варианта, используемые в шифре потока.

Упражнения

1. Блок транспозиции имеет 10 входов и 10 выходов. Каков порядок группы перестановки? Каков размер ключевой последовательности?
2. Блок подстановки имеет 10 входов и 10 выходов. Каков порядок группы перестановки? Каков размер ключевой последовательности?
3.
   - Покажите результат циркулярного левого сдвига на 3 бита на слове $(10011011)_2$.
   - Покажите результат циркулярного правого сдвига на 3 бита на слове, полученном в пункте a.
   - Сравните результат пункта b с первоначальным словом пункта a.
4.
   - Измените слово $(10011011)_2$ с помощью перестановки.
   - Измените слово, полученное по пункту a, с помощью перестановки
   - Сравните результаты пункта a и пункта b, чтобы показать, что перестановка — самообратимая операция.
5. Найдите результат следующих операций:
   - $(01001101) \oplus (01001101)$
   - $(01001101) \oplus (10110010)$
   - $(01001101) \oplus (00000000)$
   - $(01001101) \oplus (11111111)$
6.
   - Расшифруйте слово 010, используя декодер 3×8.

- Зашифруйте слово 00100000, используя кодирующее устройство 8×3.

7. Сообщение имеет 2000 символов. Оно будет зашифровано с использованием *блочного шифра* 64 битов. Найдите размер дополнения и номера блоков.
8. Покажите таблицу перестановки для прямого P -блока на рис. 7.4
9. Покажите таблицу перестановки для P -блока сжатия на рис. 7.4.
10. Покажите таблицу перестановки для P -блока расширения на рис. 7.4.
11. Покажите P -блок, определенный следующей таблицей:

    8 1 2 3 4 5 6 7

12. Определите, является ли P -блок со следующей таблицей перестановки прямым P -блоком, P -блоком сжатия или P -блоком расширения.

    1 1 2 3 4 4

13. Определите, является ли P -блок со следующей таблицей перестановки прямым P -блоком, P -блоком сжатия или P -блоком расширения.

    1 3 5 6 7

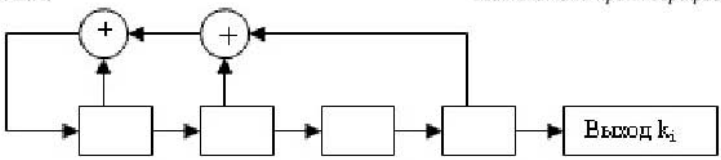
14. Определите, является ли P -блок со следующей таблицей перестановки прямым P -блоком, P -блоком сжатия или P -блоком расширения.

    1 2 3 4 5 6

15. Отношение вход-выход в 2×2 S -блока показаны в следующей таблице для S -блока. Покажите таблицу для инверсного блока.

Вход: левый бит \ Вход: правый бит	0	1
0	01	11
1	10	00

16. Покажите *LFSR* с характеристическим полиномом $x^5 + x^2 + 1$. Каков период получаемой последовательности?
17. Каков характеристический *полином* следующего *LFSR*? Каков максимальный период?



18. Покажите ключевой поток на `20` битов, сгенерированный от *LFSR* на рис. 7.25, если начальное значение — `1110`.
19. Максимальная длина периода *LFSR* — `32`. Сколько битов имеет *регистр сдвига*?
20. `6 x 2 S` -блок производит операцию ИСКЛЮЧАЮЩЕЕ ИЛИ с нечетными битами, чтобы получить левый бит выхода, и ИСКЛЮЧАЮЩЕЕ ИЛИ с четными битами, чтобы получить правый бит выхода. Если вход — `110010`, что является выходом? Если вход — `101101`, что является выходом?
21. Крайний левый бит `4 x 3 S` -блока определяет смещение других трех бит. Если крайний левый бит равен `0`, то три других бита перемещаются вправо на один бит. Если крайний левый бит — `1`, три других бита перемещаются влево на один бит. Если вход — `1011`, какой результат будет на выходе? Если вход — `0110`, какой результат будет на выходе?
22. Напишите процедуру в псевдокоде для разбиения `n` -битового слова на два слова, каждое из которых состоит из `n/2`.
23. Напишите процедуру в псевдокоде для объединения двух `n/2` -битовых слов в `n` -битовое слово.
24. Напишите процедуру в псевдокоде, которая переставляет левые и правые половины `n` -битового слова.
25. Напишите процедуру в псевдокоде, которая циклически сдвигает в `n` -разрядном слове на `k` бит влево или вправо, в соответствии с процедурой по п. 24.
26. Напишите процедуру в псевдокоде для `P` -блока, в котором перестановка определена таблицей.
27. Напишите процедуру в псевдокоде для `S` -блока, в котором вход-выход определен таблицей.
28. Напишите процедуру в псевдокоде, которая моделирует каждый раунд не-Файстеля, показанный на рис. 7.13.
29. Напишите процедуру в псевдокоде, которая моделирует каждый раунд шифра, показанный на рис. 7.17.
30. Напишите процедуру в псевдокоде, которая моделирует `n` -

битовый *LFSR*.

Стандарт шифрования данных (DES)

В этой лекции мы обсуждаем Стандарт шифрования данных (DES — DATA ENCRIPTION STANDARD) — современный блочный шифр с симметричными ключами. Наши основные цели для этой лекции: рассмотреть короткую историю DES; определить основную структуру DES; описать детали основных элементов DES; описать процесс генерации ключей для раундов; провести анализ DES. Особое внимание уделяется тому, как DES использует шифр Файстеля, чтобы достигнуть перемешивания и рассеивания на выходе из битов исходного текста к битам зашифрованного текста.

8.1. Введение

Стандарт шифрования данных (DES) — *блочный шифр* с *симметричными ключами*, разработан Национальным Институтом Стандартов и Технологии (*NIST* – National Institute of Standards and Technology).

История

В 1973 году *NIST* издал запрос для разработки предложения национальной *криптографической системы* с *симметричными ключами*.

Предложенная IBM модификация проекта, названная Lucifer, была принята как *DES*. *DES* были изданы в эскизном виде в Федеральном Регистре в марте 1975 года как Федеральный Стандарт Обработки Информации (FIPS – Federal Information Processing Standard).

После публикации эскиз строго критиковался по двум причинам. Первая: критиковалась сомнительно маленькая *длина ключа* (только 56 битов), что могло сделать шифр уязвимым к атаке "грубой силой". Вторая причина: критики были обеспокоены некоторым скрытым построением внутренней структуры *DES*.

Они подозревали, что некоторая часть структуры (S -блоки) может

иметь скрытую лазейку, которая позволит расшифровывать сообщения без ключа. Впоследствии проектировщики IBM сообщили, что внутренняя структура была доработана, чтобы предотвратить *криптоанализ*.

DES был наконец издан как *FIPS* 46 в Федеральном Регистре в январе 1977 года. Однако *FIPS* объявил *DES* как стандарт для использования в неофициальных приложениях. *DES* был наиболее широко используемым блочным шифром с *симметричными ключами*, начиная с его публикации. Позже *NIST* предложил новый стандарт (*FIPS* 46-3), который рекомендует использование *тройного DES* (трехкратно повторенный шифр *DES*) для будущих приложений. Как мы увидим далее, в лекциях 9-10, предполагается, что более новый стандарт *AES* заменит *DES*.

Общие положения

Как показано на рис. 8.1. , *DES — блочный шифр*.

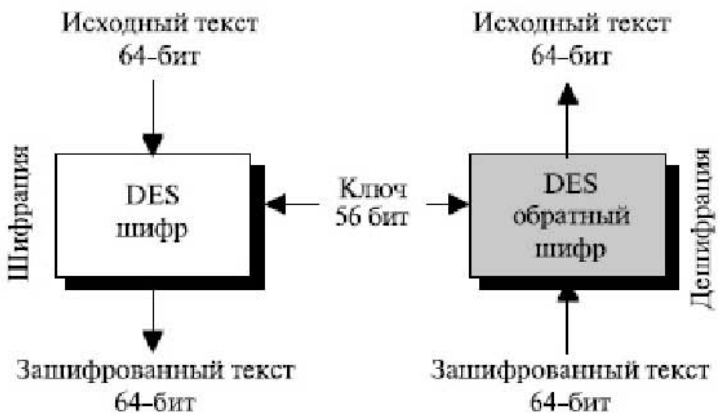


Рис. 8.1. Шифрование и дешифрование в DES

На стороне шифрования *DES* принимает 64 -битовый исходный текст и порождает 64 -битовый зашифрованный текст; на стороне дешифрования *DES* принимает 64 -битовый зашифрованный текст и порождает 64 -битовый исходный текст. На обеих сторонах для шифрования и дешифрования применяется один и тот же 56 -битовый

ключ.

8.2. Структура DES

Рассмотрим сначала шифрование, а потом дешифрование. Процесс шифрования состоит из двух перестановок (P -блоки) — они называются начальные и конечные перестановки, — и шестнадцати раундов Файстеля. Каждый раунд использует различные сгенерированные 48 -битовые ключи. Алгоритм генерации будет рассмотрен в этой лекции позднее. Рисунок 8.2 показывает элементы шифра *DES* на стороне шифрования.

Начальные и конечные перестановки

Рисунок 8.3 показывает начальные и конечные перестановки (P -блоки). Каждая из перестановок принимает 64 -битовый вход и переставляет его элементы по заданному правилу. Мы показали только небольшое число входных портов и соответствующих выходных портов. Эти перестановки — прямые перестановки без ключей, которые инверсны друг другу. Например, в начальной перестановке 58 -й бит на входе переходит в первый бит на выходе. Аналогично, в конечной перестановке первый входной бит переходит в 58 -й бит на выходе. Другими словами, если между этими двумя перестановками не существует раунда, 58 -й бит, поступивший на вход устройства начальной перестановки, будет доставлен на 58 -й выход финальной перестановкой.

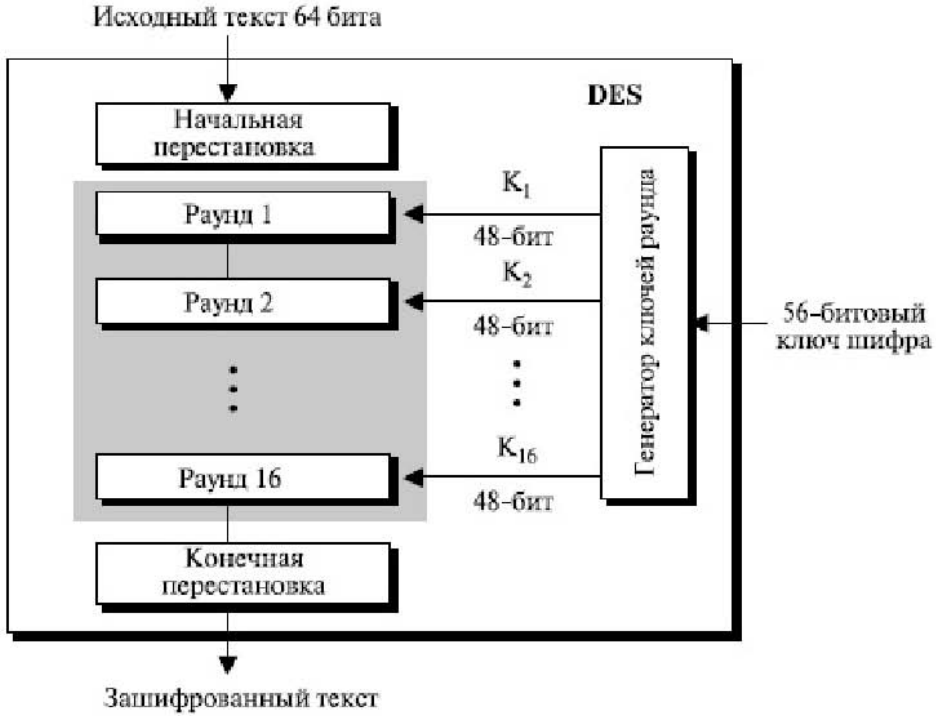


Рис. 8.2. Общая структура DES

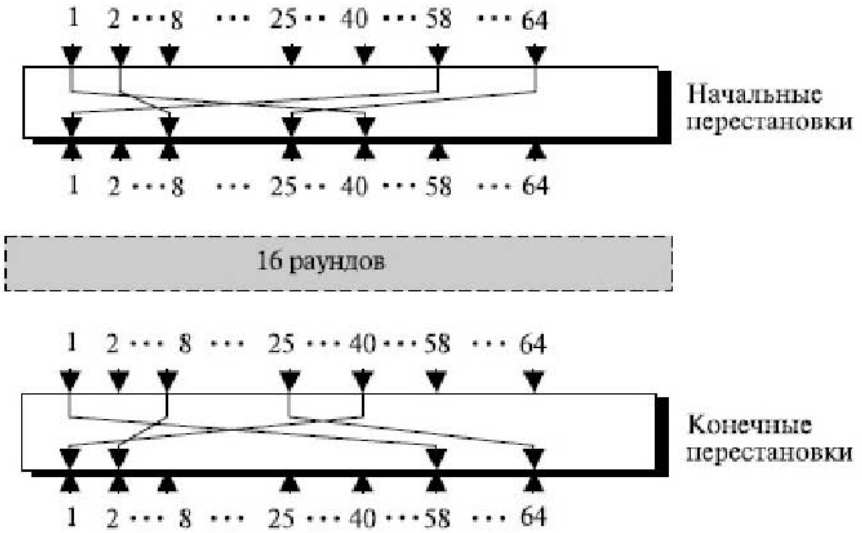


Рис. 8.3. Начальные и конечные шаги перестановки DES

Правила перестановки для этого `P` -блока показаны в таблице 8.1. Таблицу можно представить как `64` -элементный массив. Заметим, что работу с таблицей мы обсуждали, значение каждого элемента определяет номер входного порта, а порядковый номер (индекс) элемента определяет номер выходного порта.

Таблица 8.1. Таблица начальных и конечных перестановок

Начальные перестановки								Конечные перестановки							
58	50	42	34	26	18	10	02	40	08	48	16	56	24	64	32
60	52	44	36	28	20	12	04	39	07	47	15	55	23	63	31
62	54	46	38	30	22	14	06	38	06	46	14	54	22	62	30
64	56	48	40	32	24	16	08	37	05	45	13	53	21	61	29
57	49	41	33	25	17	09	01	36	04	44	12	52	20	60	28
59	51	43	35	27	19	11	03	35	03	43	11	51	19	59	27
61	53	45	37	29	21	13	05	34	02	42	10	50	18	58	26
63	55	47	39	31	23	15	07	33	01	41	09	49	17	57	25

Эти две перестановки не имеют никакого значения для криптографии в *DES*. Обе перестановки – без ключей и предопределенны. Причина, почему они включены в *DES*, не ясна и не была указана проектировщиками *DES*. Можно предположить, что *DES* был проектом, который предполагалось реализовать в аппаратных средствах (на чипах), и что эти две сложные перестановки должны были затруднить программное моделирование механизма шифрования.

Пример 8.1

Найдите выход начального блока перестановки, когда на вход поступает шестнадцатеричная последовательность, такая как

0x0002 0000 0000 0001

Решение

Вход имеет только две единицы — (бит `15` и бит `64`); выход должен также иметь только две единицы (прямая перестановка). Используя

таблицу 8.1, мы можем найти выход, связанный с этими двумя битами. Бит 15 на входе становится битом 63 в выходе. Бит 64 во входе становится битом 25 в выходе. На выходе будем иметь только две единицы — бит 25 и бит 63.

Результат в шестнадцатеричном исчислении

0x0000 0080 0000 0002

Пример 8.2

Докажем, что начальные и финальные перестановки инверсны друг другу. Преобразуем полученную выходную последовательность во входную.

0x0000 0080 0000 0002

Решение

Единичные биты — 25 и 63, другие биты равны нулю. В конечной перестановке 25 -й бит переходит в 64 -й, а 63 -й — в 15 -й. Результат

0x0002000000000001

Начальные и конечные перестановки – это прямые P -блоки, которые инверсны друг другу. Они не имеют значения для криптографии DES.

Раунды

DES использует 16 раундов. Каждый раунд *DES* применяет шифр Файстеля, как это показано на рис. 8.4.

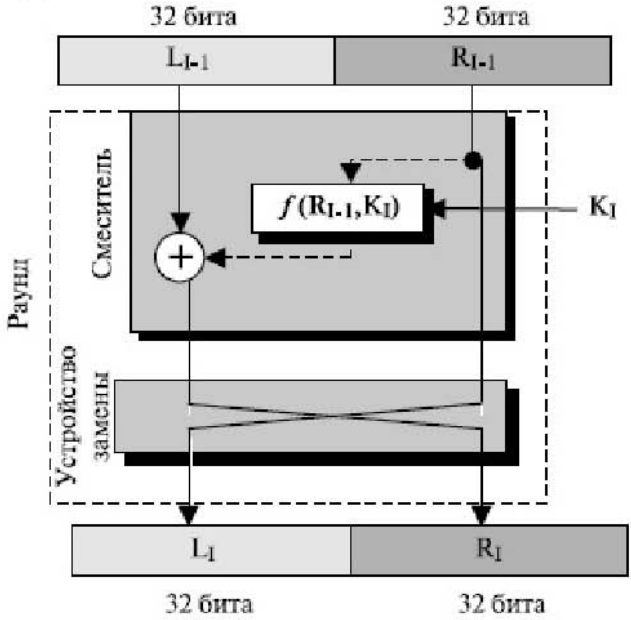


Рис. 8.4. Раунд в DES (сторона шифрования)

Раунд принимает L_{I-1} R_{I-1} от предыдущего раунда (или начального блока перестановки) и создает для следующего раунда L_I И R_I, которые поступают на следующий раунд (или конечный блок перестановки). Как мы указали в лекции 7, можно принять, что каждый раунд имеет два элемента шифра (смеситель и устройство замены). Каждый из этих элементов является обратимым. Устройство замены — очевидно обратимо, оно меняет местами левую половину текста с правой половиной. Смеситель является обратимым, потому что операция *ИСКЛЮЧАЮШЕЕ ИЛИ* обратима. Все *необратимые* элементы сосредоточены в функции $f(R_{I-1},K_I)$.

Функция DES

Основной блок *DES* — функция *DES*. Функция *DES* с помощью 48 -битового ключа зашифровывает 32 самых правых бит R_{I-1}, чтобы получить на выходе 32 -битовое слово. Эта функция содержит, как это показано на рис. 8.5, четыре секции: отбеливатель (*whitener*), P -блок расширения, группу S -блоков и прямой P -блок.

Р-блок расширения. Так как вход R_{I-1} имеет длину 32 бита, а ключ K_I — длину 48 битов, мы сначала должны расширить R_{I-1} до 48 бит. R_{I-1} разделяется на 8 секций по 4 бита. Каждая секция на 4 бита расширяется до 6 бит. Эта перестановка расширения следует по заранее определенным правилам. Для секции значения входных бит 1, 2, 3 и 4 присваиваются битам 2, 3, 4 и 5 соответственно на выходе. Выходной бит 1 формируется на основе входного бита 4 из предыдущей секции; бит выхода 6 формируется из бита 1 в следующей секции. Если секции 1 и 8 рассматривать как соседние секции, то те же самые правила применяются к битам 1 и 32. Рисунок 8.6 показывает входы и выходы в перестановке расширения.

Хотя отношения между входом и выходом могут быть определены математически, но чтобы определить этот P -блок, *DES* использует таблицу 8.2. Обратите внимание, что число выходов 48, но диапазон значений — только от 1 до 32. Некоторые из входов идут к больше чем одному выходу. Например, значение входного бита 5 становится значением битов выхода 6 и 8.

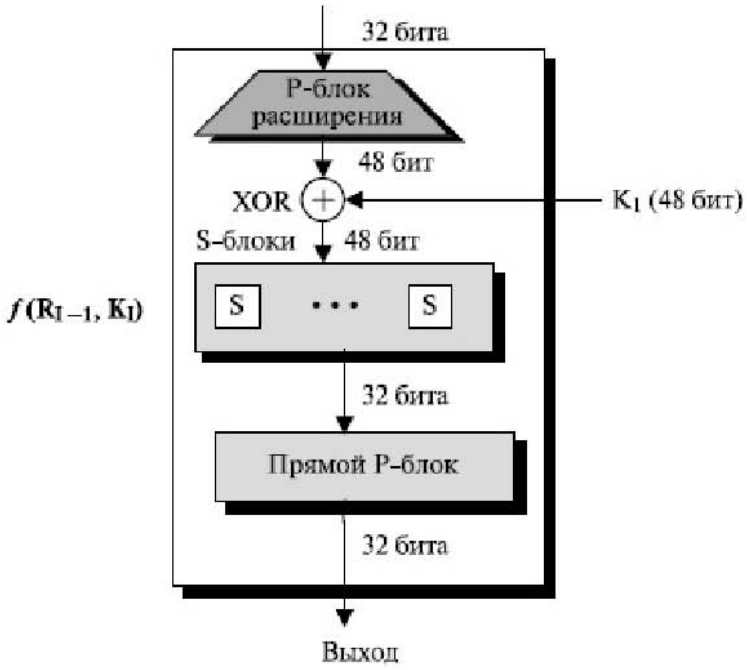


Рис. 8.5. Функция DES

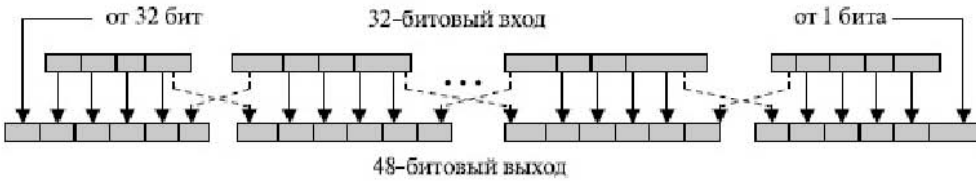


Рис. 8.6. Перестановка расширения

Таблица 8.2.
Таблица P-блока расширения

32	01	02	03	04	05
04	05	06	07	08	09
08	09	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	31	31	32	01

Отбеливатель (whitener). После расширения *DES* использует операцию `XOR` (*ИСКЛЮЧАЮШЕЕ ИЛИ*) над расширенной частью правой секции и ключом раунда. Заметим, что правая секция и ключ имеют длину `48` бит. Также заметим, что ключ раунда использует только эту операцию.

S-блоки. `S` -блоки смешивают информацию (операция перемешивания). *DES* использует `S` -блоки, каждый с `6` -ю входными битами и `4` -мя выходными (см. рис. 8.7).

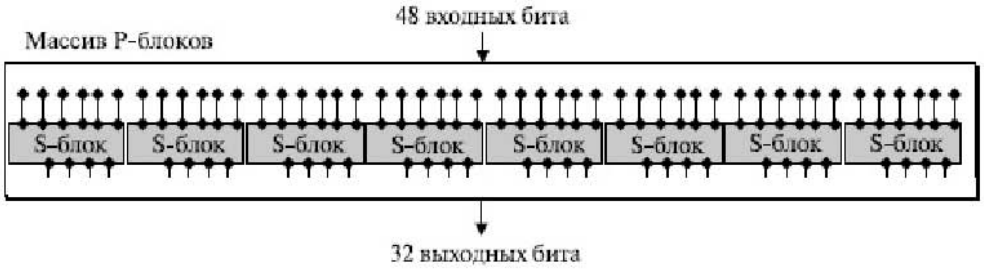


Рис. 8.7. S - блоки

Данные из 48 битов от второй операции *DES* разделены на восемь кусков по 6 битов, и каждый кусок поступает в блок. Результат каждого блока — кусок на 4 бита; когда они объединены, результат выражается в 32 -битовом тексте. Подстановка в каждом блоке следует по заранее определенным правилам, основанным на таблице из 4 -х строк и 16 -ти столбцов. Комбинация битов 1 и 6 на входе определяет одну из четырех строк; комбинация битов от 2 -го до 5 -го определяет один из шестнадцати столбцов, как показано на рис. 8.8. Далее мы поясним это примерами.

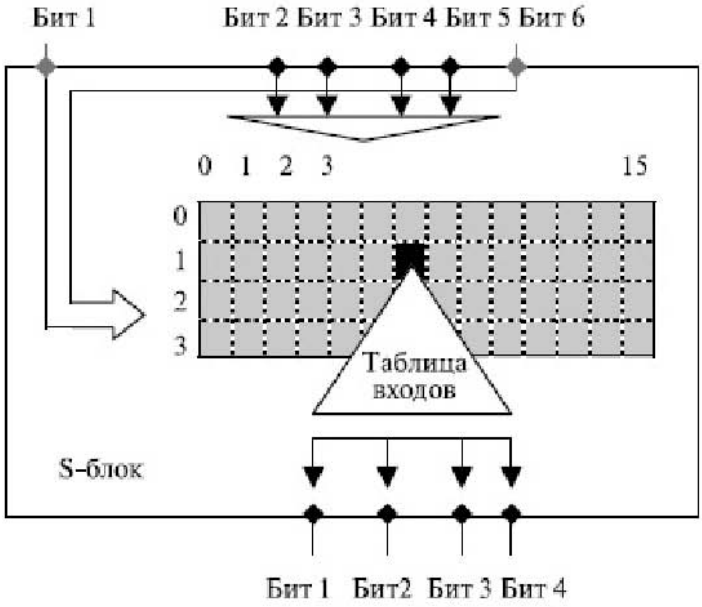


Рис. 8.8. Правила для S- блока

Поскольку для каждого S -блока есть собственная таблица, необходимо иметь восемь таблиц, например таких, как это показано в таблицах 8.3-8.10. Значение входа (номер строки и номер столбца) и значения выхода даются как десятичные номера, чтобы сэкономить место на странице. В реальности они могут быть заменены двоичными числами.

Таблица 8.3. S-блок 1

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	14	04	13	01	02	15	11	08	03	10	06	12	05	09	00	07
1	00	15	07	04	14	02	13	10	03	06	12	11	09	05	03	08
2	04	01	14	07	13	06	02	11	15	12	09	07	03	10	05	00
3	15	12	08	02	04	09	01	07	05	11	03	14	10	00	06	13

Таблица 8.4. S-блок 2

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	15	01	08	14	06	11	03	04	09	07	02	13	12	00	05	10
1	03	13	04	07	15	02	08	14	12	00	01	10	06	09	11	05
2	00	14	07	11	10	04	13	01	05	08	12	06	09	03	02	15
3	13	08	10	01	03	15	04	02	11	06	07	12	00	05	14	09

Таблица 8.5. S-блок 3

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	10	00	09	14	06	03	15	05	01	13	12	07	11	04	02	08
1	13	07	00	09	03	04	06	10	02	08	05	14	12	11	15	01
2	13	06	04	09	08	15	03	00	11	01	02	12	05	10	14	07
3	01	10	13	00	06	09	08	07	04	15	14	03	11	05	02	12

Таблица 8.6. S-блок 4

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	07	13	14	03	00	06	09	10	01	02	08	05	01	12	04	15
1	13	08	11	05	06	15	00	03	04	07	02	12	01	10	14	09
2	10	06	09	00	12	11	07	13	15	01	03	14	05	02	08	04
3	03	15	00	06	10	01	13	08	09	04	05	11	12	07	02	14

Таблица 8.7. S-блок 5

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

0	02	12	04	01	07	10	11	06	08	05	03	15	13	00	14	09
1	14	11	02	12	04	07	13	01	05	00	15	10	03	09	08	06
2	04	02	01	11	10	13	07	08	15	09	12	05	06	03	00	14
3	11	08	12	07	01	14	02	13	06	15	00	09	10	04	05	03

Таблица 8.8. S-блок 6

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	12	01	10	15	09	02	06	08	00	13	03	04	14	07	05	11
1	10	15	04	02	07	12	09	05	06	01	13	14	00	11	03	08
2	09	14	15	05	02	08	12	03	07	00	04	10	01	13	11	06
3	04	03	02	12	09	05	15	10	11	14	01	07	10	00	08	13

Таблица 8.9. S-блок 7

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	04	11	02	14	15	00	08	13	03	12	09	07	05	10	06	01
1	13	00	11	07	04	09	01	10	14	03	05	12	02	15	08	06
2	01	04	11	13	12	03	07	14	10	15	06	08	00	05	09	02
3	06	11	13	08	01	04	10	07	09	05	00	15	14	02	03	12

Таблица 8.10. S-блок 8

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	13	02	08	04	06	15	11	01	10	09	03	14	05	00	12	07
1	01	15	13	08	10	03	07	04	12	05	06	11	10	14	09	02
2	07	11	04	01	09	12	14	02	00	06	10	10	15	03	05	08
3	02	01	14	07	04	10	05	13	15	19	09	09	03	05	06	11

Пример 8.3

Входная последовательность S блока 1 — 100011. Какая последовательность будет на выходе?

Решение

Если мы запишем первый и шестой биты вместе, мы получим в двоичном исчислении 11, который выражается как число 3 при десятичном исчислении. Остающаяся часть битов 0001 в двоичном исчислении является 1 в десятичном исчислении. Мы ищем значение

строки 3 и столбца 1 в таблице 8.3 (S -блок 1). Результат — 12 в десятичном исчислении или 1100 в двоичном исчислении. Тогда вход 1100011 дает выход 1100.

Пример 8.4

Входная последовательность S -блока 8 — 000000. Какая последовательность будет на выходе?

Решение

Если мы запишем первый и шестые биты вместе, то получим в двоичном исчислении число 00, которое выражается числом 0 при десятичном исчислении. Остающаяся часть битов — 0000 в двоичном исчислении, то есть 0 в десятичном исчислении. Мы ищем значение строки 0 и столбца 0 в таблице 8.10 (S -блок 8). Результат — 13 в десятичном исчислении или 1101 в двоичном исчислении. Тогда вход 000000 дает выход 1101.

Прямая перестановка — последняя операция в функции *DES* — прямая перестановка с 32 битами на входе и 32 битами на выходе. Отношения "вход-выход" для этой операции показаны в Таблице 8.11. Они следуют тем же самым общим правилам, как и предыдущие таблицы перестановки. Например, седьмой бит входа становится вторым битом выхода.

Таблица 8.11. Таблица прямой перестановки

16	07	20	21	29	12	28	17
01	15	23	26	05	18	31	10
02	08	24	14	32	27	03	09
19	13	30	06	22	11	04	25

Шифр и обратный шифр

Используя смеситель и устройство замены, мы можем создать шифр и

обратный шифр для каждого из 16 -ти раундов. Шифр используется на стороне шифрования; обратный шифр — на стороне дешифрования. Алгоритмы создания шифра и обратного шифра аналогичны.

Первый способ

Один из методов, чтобы достигнуть поставленной цели (шифрование и обратное шифрование), состоит в том, чтобы сделать последний раунд отличающимся от других; он будет содержать только смеситель и не будет содержать устройства замены, как это показано на рис. 8.9.

Мы доказали в лекции 7, что смеситель и устройство замены самоинверсны. Конечные и начальные перестановки также инверсны друг другу. Левая секция исходного текста на стороне шифрования шифруется L_0 как L_{16}, и L_{16} дешифруется на стороне *дешифратора* как L_0. Аналогичная ситуация с R_0 и R_{16}.

Нужно запомнить очень важное положение, которое касается шифров: ключи раундов (K_1 и K_{16}) применяются при шифровании и дешифровании в обратном порядке. На стороне шифрования первый раунд применяет ключ K_1, а раунд 16 — ключ K_{16} ; при дешифровании раунд 1 использует ключ K_{16}, а раунд 16 — ключ K_1.

В первом методе последний раунд не имеет устройства замены.

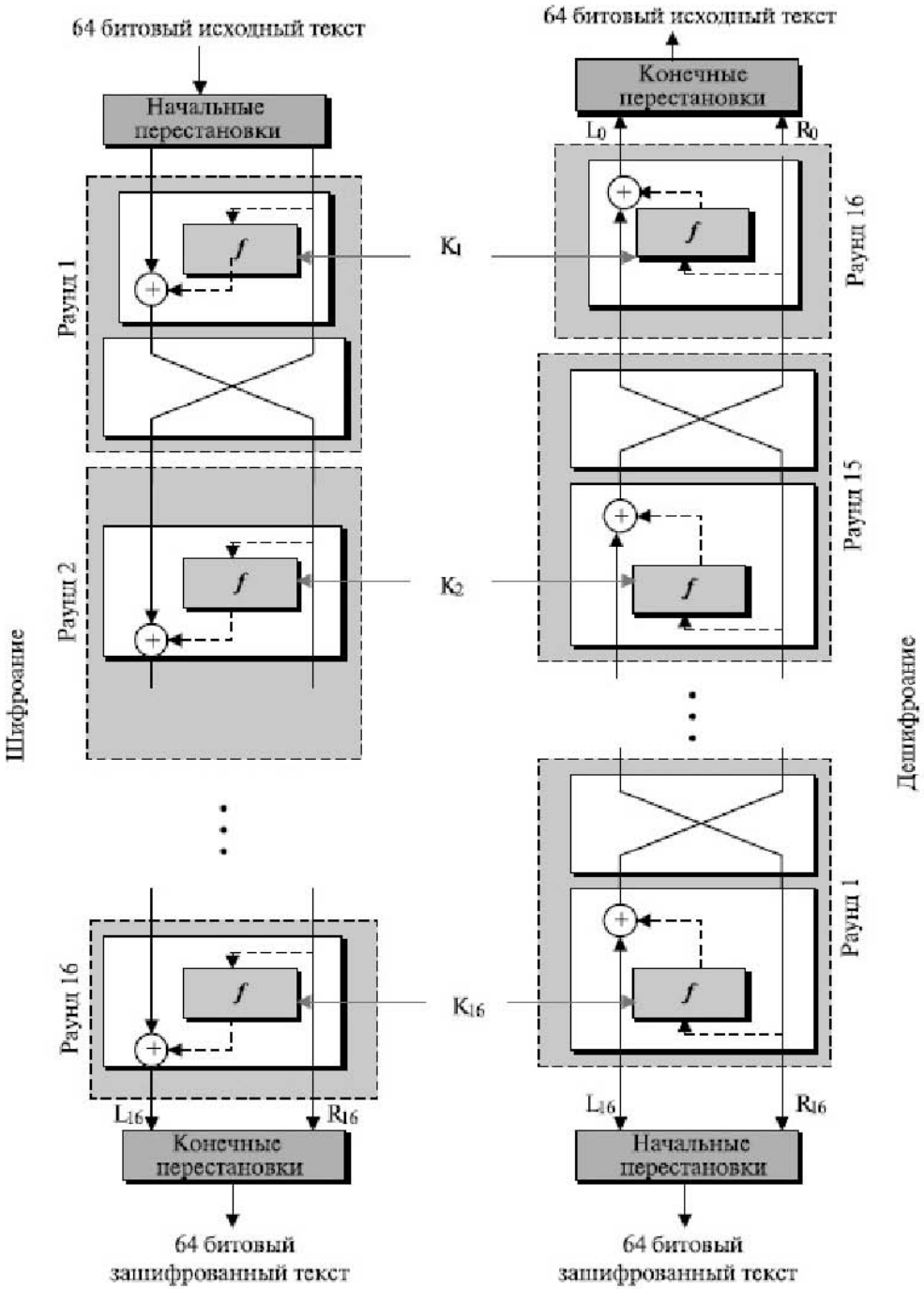


Рис. 8.9. DES шифр и обратный шифр для первого способа

Алгоритм

Алгоритм 8.1. приведен в *псевдокодах* для шифрования и соответствует четырем шагам первого метода. Коды для остальных могут быть сделаны как упражнение.

```
Cipher (plainBlock[64], Round Keys[16,48])
{
  permute (64,64, plainBlock, inBlock, IntialPermutationTable)
  spilt (64, 32, inBlock, leftBlock, right Block)
  for (round = 1 to 16)
  {
    mixer (leftBlock, right Block, RoundKey[round])
    if (round!=16) swapper(leftBlock, right Block)
  }
  combine (32, 64, leftBlock, right Block, outBlock)
  permute (64,64, outBlock, cipherBlock, FinalPermutationTable)

mixer (leftBlock[48], right Block[48], RoundKey[48])
{
  copy (32, rightBlock, T1)
  function (T1, RoundKey,T2)
  exclusive Or (32, leftBlock, T2,T3)
  copy (32, rightBlock, T1)
}

swapper (leftBlock[32], right Block[32])
{
  copy (32, leftBlock, T)
  copy (32, rightBlock, leftBlock)
  copy (32, T, rightBlock)
}

Substitute (in block[32], outblock[48], SubstituteTables[8,4,16])
{
  for(I = 1 to 8)
  {
    row <- 2 x inBlock[i x 6+1] + inBlock[i x 6+6]
    col <- 8 x inBlock[i x 6+2] + 4 x inBlock[i x 6+3]+2 x inBlock[i x 6+4] + in
```

```
        value = SubstituteTables[i][row][col]

        outBlock[i x 4+1] <- value/ 8        value <- value mod 8
        outBlock[i x 4+2] <- value/ 4        value <- value mod 4
        outBlock[i x 4+3] <- value/ 2        value <- value mod 8
        outBlock[i x 4+4] <- value
    }
  }
  }
```

Пример 8.1. Псевдокоды для DES шифра

Альтернативный способ

При первом способе раунд 16 отличается от других раундов тем, что там не применяется устройство замены. Это необходимо, чтобы сделать последний и первый смесители в шифре одинаковыми. Мы можем делать все 16 раундов одинаковыми, добавляя к 16 -му раунду дополнительное устройство замены (два устройства замены позволяют нейтрализовать друг друга). Разработку этой схемы мы оставляем для упражнений.

Генерация ключей

Генератор ключей создает шестнадцать ключей по 48 битов из ключа шифра на 56 битов. Однако ключ шифра обычно дается как ключ из 64 -х битов, в котором 8 дополнительных битов являются битами проверки. Они отбрасываются перед фактическим процессом *генерации ключей*, который показан на рис. 8. 10.

Удаление битов проверки

Предварительный процесс перед расширением ключей — перестановка сжатия, которую мы называем удалением битов проверки. Он удаляет биты четности (биты 8, 16, 24, 32..., 64) из 64 -битового

ключа и переставляет остальную часть битов согласно таблице 8.12. Остающееся значение на 56 битов — фактический ключ шифра, который используется, чтобы генерировать ключи раунда. Биты удаляются с помощью перестановки (P -блока сжатия), как это показано в таблице 8.12.

Таблица 8.12. Таблица удаления проверочных битов

57	49	41	33	25	17	09	01
58	50	42	34	26	18	10	02
59	51	43	35	27	19	11	03
60	52	44	36	63	55	47	39
31	23	15	07	62	54	46	37
30	22	14	06	61	53	45	37
29	21	13	05	28	20	12	04

Сдвиг

Раунд	Сдвиг
1,2,9,16	один бит
Остальные	два бита

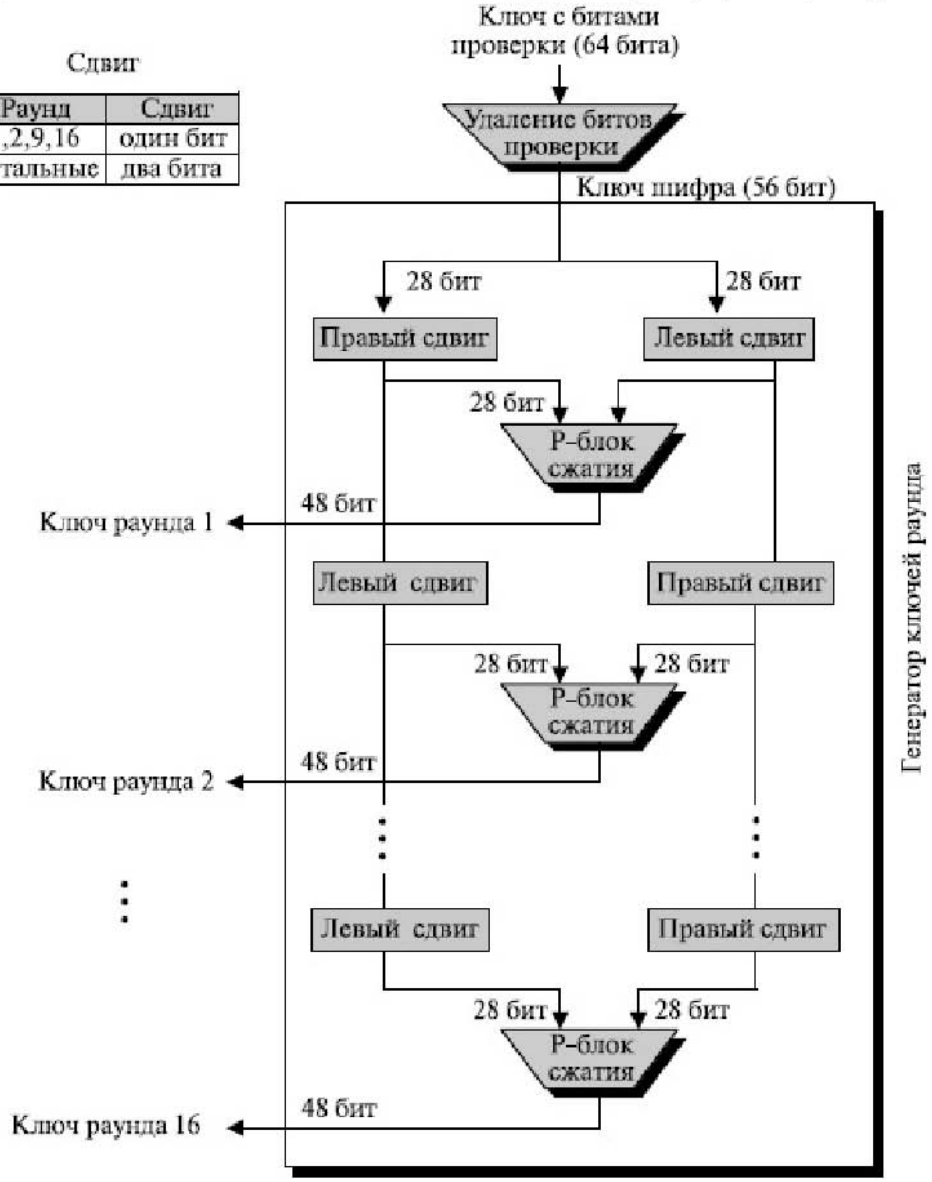


Рис. 8.10. Генерация ключей

Сдвиг влево

После прямой перестановки ключ разделен на две части по 28 битов. Каждая часть сдвигается влево (циклический сдвиг) на один или два

бита. В раундах 1, 2, 9 и 16 смещение – на один бит, в других раундах — на два бита. Затем эти две части объединяются, чтобы создать часть в 56 бит. Таблица 8.13 показывает число сдвигов манипуляций для каждого раунда.

Таблица 8.13. Число сдвигаемых бит

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

Перестановка сжатия

Перестановка сжатия (P -блок) изменяет 56 битов на 48 битов, которые используются для формирования ключа раунда. Перестановка сжатия показана в таблице 8.14.

Таблица 8.14. Таблица сжатия ключа

14	17	11	24	01	05	03	28
15	06	21	10	23	19	12	04
26	08	16	07	27	20	13	02
41	52	31	37	47	55	30	40
51	45	33	48	44	49	39	56
34	53	46	42	50	36	29	32

Алгоритм

Теперь напишем простой алгоритм для создания ключа из ключа с проверочными битами. Алгоритм 8.2 использует несколько процедур алгоритма 8.1. Имеется и одна новая процедура — левый сдвиг (ShiftLeft). Обратите внимание, что T — временный блок.

```
Key_Generator (keyWithParities[64], RoundKeys[16. 48]. SinfTable[16])
 {                                          '
  permute (64, 56, keyWithParities, cipherKey, ParityDropTable)
 spilt (56, 28, cipherKey, leftKey, rightKey)
```

```
  for (round = 1 to 16)
    {
    shiftLeft (leftKey, ShiftTable[round1])                    I
    shiftLeft (rightKey, ShiftTable [round])
    combine (28, 56, leftKey, rightKey, preRoundKey)
    permute (56, 48, preRoundKey, RoundKeys[round], KevCompressionTable)
    }
  }

  shiftLeft (block[28], NumOfShifts)
  {
    for (i = 1 to numShifts)
    {
      T <- block[ 1]
      for (j = 2 to 28)
      {
         block [j-1] <- block [j]
      }
      block[28] <- T
    }
  }
```

Пример 8.2. Алгоритм 8.2. Алгоритм для генерации ключей раунда

Примеры

Перед анализом *DES* рассмотрим несколько примеров, чтобы понять, как шифрование и дешифрование меняют значение битов в каждом раунде.

Пример 8.5

Мы выбираем случайный блок исходного текста и случайный ключ и определяем, каким должен быть блок зашифрованного текста (все цифры даны в шестнадцатеричном исчислении).

Plaintext: 123456ABCD 132536 Key: AABB09182736CCDD
Cipher Text: COB7ASD05F3AS29C

Покажем результат каждого раунда и текста, созданного до и после раундов. Таблица 8.15 показывает результаты первых шагов перед началом раунда. Исходный текст — прошедший через начальную перестановку для получения различных 64 бит (16 шестнадцатеричных цифр).

Таблица 8.15. Трассировка данных в примере 8.5

Исходный текст: 123456ABCD 132536			
После первоначальной перестановки: 14A7D67818CA18D После разбиения: L_0= 14A7D678 R_0 = 18CA18D			
Раунд	Левая	Правая	Ключ раунда
Раунд 1	18CA18AD	5A78E394	194CD072DE8C
Раунд 2	5A78E394	4A1210F6	4568581ABCCE
Раунд 3	4A1210F6	B8089591	06EDA4ACF5B5
Раунд 4	B8089591	236779C2	DA2D032B6EE3
Раунд 5	236779C2	A15A4B87	69A629FEC913
Раунд 6	A15A4B87	2E8F9C65	C1948E87475E
Раунд 7	2E8F9C65	A9FC20A3	708AD2DDB3C0
Раунд 8	A9FC20A3	308BEE97	34F822F0C66D
Раунд 9	308BEE97	10AF9D37	84BB4473DCCC
Раунд 10	10AF9D37	6CA6CB20	02765708B5BF
Раунд 11	6CA6CB20	FF3C485F	6D5560AF7CA5
Раунд 12	FF3C485F	22A5963B	C2C1E96A4BF3
Раунд 13	22A5963B	387CCDAA	99C31397C91F
Раунд 14	387CCDAA	BD2DD2AB	251B8BC717D0
Раунд 15	BD2DD2AB	CF26B472	3330C5D9A36D
Раунд 16	19BA9212	CF26B472	181C5D75C66D
После объединения: 19BA9212 CF26B472			
Зашифрованный текст: C0B7A8D05F3A829C		(после конечной перестановки)	

Таблица показывает результат 16 раундов, которые включают смешивание и замену (исключая последний раунд). Результаты

последних раундов (L_{16} и R_{16}) объединены. Наконец, текст проходит конечную перестановку, для того чтобы получить зашифрованный текст.

Следует отметить некоторые положения. Правая секция каждого раунда совпадает с левой секцией следующего раунда. Причина в том, что правая секция проходит через смеситель без изменения, а устройство замены переносит ее в левую секцию. Например, R_1 передается через смеситель второго раунда без изменения, но затем, пройдя устройство замены, он становится L_2. Второе интересное положение: на последнем раунде мы не имеем устройства замены. Именно поэтому R_{15} становится R_{16} вместо того чтобы стать L_{16}.

Пример 8.6

Давайте рассмотрим, как Боб в пункте назначения может расшифровать зашифрованный текст, полученный от Алисы, с помощью совпадающего ключа. Для экономии времени мы разберем только несколько раундов. Таблица 8.16 показывает интересующие нас точки. Первое правило: ключи раунда должны использоваться в обратном порядке. Сравните таблицу 8.15 и таблицу 8.16. Ключ раунда 1 — такой же как ключ для раунда 16. Значения L_0 и R_0 при дешифрации те же самые, что и значения L_{16} и R_{16} при шифровании. Аналогичные совпадения будут получены и в других раундах. Это доказывает не только, что шифр и обратный шифр инверсны друг другу, но также то, что каждый раунд при шифрации имеет соответствующий раунд при дешифрации в обратном шифре. Результат свидетельствует, что начальные и конечные перестановки также являются инверсиями друг друга.

Таблица 8.16. Трассировка данных в примере 8.6

Зашифрованный текст: C0B7A8D05F3A829C			
После первоначальной перестановки: 19BA9212 CF26B472 После разбиения: L_0= 19BA9212 R_0 = CF26B472			
Раунд	Левая	Правая	Ключ раунда
Раунд 1	CF26B472	BD2DD2AB	181C5D75C66D
Раунд 2	BD2DD2AB	387CCDAA	3330C5D9A36D

.....	.....	.....	.....
Раунд 15	5A78E394	18CA182AD	4568581ABCCE
Раунд 16	19BA9212	18CA18AD	194CD072DE8C
После объединения: 14A7D67818CA18D			
Исходный текст: 123456ABCD 132536		(после конечной перестановки)	

8.3. Анализ DES

DES был подвергнут тщательному анализу. Были проведены испытания, чтобы измерить интенсивность некоторых желательных свойств в *блочном шифре*. Элементы *DES* прошли исследования на соответствие некоторым критериям. Ниже мы обсудим некоторые из них.

Свойства

Два желательных свойства блочного шифра — эффект лавины и законченность.

Лавинный эффект

Лавинный эффект означает, что небольшие изменения в исходном тексте (или ключе) могут вызвать значительные изменения в зашифрованном тексте. Было доказано, что *DES* имеет все признаки этого свойства.

Пример 8.7

Чтобы проверить эффект лавины в *DES*, попробуем зашифровать два блока исходного текста, которые отличаются только одним битом текста, с помощью одного и того же ключа и определим разницу в числе бит в каждом раунде.

Исходный текст: 0000000000000000 Ключ: 22234512987ABB23
Зашифрованный текст: 4789FD476E82A5F1

Исходный текст: 0000000000000001 Ключ: 22234512987ABB23
Зашифрованный текст: OA4ED5C15A63FEA3

Хотя два блока исходного текста отличаются только самым правым битом, блоки зашифрованного текста отличаются на 29 бит. Это означает, что изменение приблизительно в 1,5 процентах исходного текста создают изменение приблизительно 45 процентов зашифрованного текста. Таблица 8.17 показывает изменение в каждом раунде. Можно увидеть, что существенные изменения возникают уже в третьем раунде.

Таблица 8.17. Число различных бит в примере 8.7

Раунд	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Разница в битах	1	6	20	29	30	33	32	29	32	39	33	28	30	31	30	29

Эффект полноты

Эффект полноты заключается в том, что каждый бит зашифрованного текста должен зависеть от многих битов исходного текста. Рассеивание и перемешивание, произведенное P -блоками и S -блоками в *DES*, указывает на очень сильный эффект полноты.

Критерии разработок DES

Проект *DES* был предъявлен IBM в 1994 году. Многочисленные испытания *DES* показали, что он удовлетворяет некоторым из заявленных критериев. Ниже кратко обсуждаются некоторые проблемы разработок *DES*.

S-блоки

Мы обсудили *общие критерии* построения S -блоков в лекции 7. Здесь мы только обсуждаем критерии, выбранные для *DES*. Структура блоков обеспечивает перемешивание и рассеивание от каждого раунда до следующего. Согласно этому положению и некоторому анализу, мы

можем упомянуть несколько свойств S -блоков.

1. Входы каждой строки есть перестановки значений между 0 и 15.
2. S -блоки — нелинейные. Другими словами, выход — не аффинное преобразование. (См. лекции 4 (аффинное преобразование) и 5, где рассматривалась линейность и S -блоков.)
3. Если мы изменяем единственный бит на входе, на выходе будут изменены два или больше бита.
4. Если два входа S -блока отличаются только двумя средними битами (битами 3 и 4), *выходная информация* должна отличаться по крайней мере двумя битами. Другими словами, S (x) и $S(x \oplus 001100)$ должны отличаться по крайней мере двумя битами, где x — вход и S (x) — выход.
5. Если два входа в S -блок отличаются первыми двумя битами (биты 1 и 2) и последними двумя битами (5 и 6), два выхода должны быть различны. Другими словами, мы должны иметь следующее отношение: $S(x) \neq S(x \oplus 11bc00)$, в котором b и c — произвольные биты.
6. Есть только 32 шестибитовые пары "вход-выход" (x_i и x_j), в которых $x_i \oplus x_j \neq (000000)_2$. Эти 32 входных пары создают 32 пары слова выхода по 4 бита. Если мы создаем какие-то различия между 32 выходами пар, $d = y_i \oplus y_j$, то из этих d должны быть одинаковыми не больше чем 8.
7. Такой же критерий, как в пункте 6, применяется к трем S -блокам.
8. В любом S -блоке, если единственный входной бит сохраняется как константа (0 или 1), то другие биты изменяются случайно так, чтобы разности между числом нулей и единиц были минимизированы.

P-блоки

Между двумя рядами S -блоков (в двух последующих раундах), есть один прямой P -блок (32 на 32) и один P -блок расширения (32 на 48). Эти два P -блока вместе обеспечивают рассеивание битов. Мы уже

говорили об общем принципе построения P -блока в лекции 7. Здесь мы обсудим только прикладные P -блоки, используемые в *DES*. В структуре P -блоков были реализованы следующие критерии:

1. Каждый вход S -блока подключается к выходу другого S -блока (в предыдущем раунде).
2. Ни один вход к данному S -блоку не соединяется с выходом от того же самого блока (в предыдущем раунде).
3. Четыре бита от каждого S -блока идут в шесть различных S -блоков (в следующем раунде).
4. Ни один из двух битов выхода от S -блока не идет в тот же самый S -блок (в следующем раунде).
5. Если число блоков S -блоков 8, то S_1, S_2, . . . , S_8.
    - Выход S_{j-2} переходит в один из первых двух битов S_j (в следующем раунде).
    - Бит выхода от S_{i-1} переходит в один из последних двух битов S_j (в следующем раунде).
    - Выход S_{j+1} переходит в один из двух средних битов S_j (в следующем раунде).
6. Для каждого S -блока два бита выхода идут в первые или последние два бита S -блока в следующем раунде. Другие два бита выхода идут в средние биты S -блока в следующем раунде.
7. Если выход от S_j переходит в один из средних битов в S_k (в следующем раунде), то бит выхода от S_k не может идти в средний бит S_j. Если мы допускаем $j = k$, то подразумеваем, что ни один средний бит S -блока не может идти в один из средних битов того же самого S -блока в следующем раунде.

Число раундов

DES используют шестнадцать раундов шифра Файстеля. Доказано, что после того как каждый текст зашифрован восемь раундов, каждый бит зашифрованного текста — функция каждого бита исходного текста и каждого ключевого бита. Зашифрованный текст — полностью случайная функция исходного текста и зашифрованного текста. Отсюда

вроде бы следует, что восьми раундов должно быть достаточно для хорошего шифрования. Однако эксперименты показывают, что некоторые версии *DES* с менее чем шестнадцатью раундами более уязвимы к атакам знания исходного текста, чем к атаке грубой силы, которая требует использования шестнадцати раундов *DES*.

Слабости DES

В течение прошлых нескольких лет критики нашли некоторые слабости в *DES*.

Мы кратко укажем на некоторые слабости, которые были обнаружены в структуре шифра.

S-блоки. В литературе указываются по крайней мере три проблемы `S` -блоков.

1. В `S` -блоке `4` три бита выхода могут быть получены тем же самым способом, что и первый бит выхода: дополнением некоторых из входных битов.
2. Два специально выбранных входа к массиву `S` -блока могут создать тот же самый выход.
3. Можно получить тот же самый выход в одном единственном раунде, изменяя биты только в трех соседних `S` -блоках.

P-блоки. В структуре `P` -блока были найдены одна загадка и одна слабость.

1. Не ясно, почему проектировщики *DES* использовали начальную и конечную перестановки. Эти перестановки не вносят никаких новых свойств с точки зрения безопасности.
2. В перестановке расширения (в функции) первые и четвертые биты последовательностей на `4` бита повторяются.

Слабость в ключе шифра

Размер ключа. Критики утверждают, что самая серьезная слабость *DES* — это размер ключа (56 битов). Чтобы предпринять атаку грубой силы данного блока зашифрованного текста, злоумышленники должны проверить 2^{56} ключей.

a. Используя доступную сегодня технологию, можно проверить один миллион ключей в секунду. Это означает, что потребуется более чем две тысячи лет, чтобы выполнить атаку грубой силы на *DES*, используя компьютер только с одним процессором.

b. Если мы можем сделать компьютер с одним миллионом чипов процессоров (параллельная обработка), то сможем проверить все множество ключей приблизительно за 20 часов. Когда был введен *DES*, стоимость такого компьютера была более чем несколько миллионов долларов, но она быстро снизилась. Специальный компьютер был создан в 1998 году — и нашел ключ за 112 часов.

c. Компьютерные сети могут моделировать *параллельную обработку*. В 1977 году команда исследователей использовала 3500 компьютеров, подключенных к Internet, чтобы найти ключ *RSA* за 120 дней. Множество ключей было разделено среди всех этих компьютеров, и каждый компьютер был ответственен за проверку части домена *DES*. Если 3500 связанных в сеть компьютеров могут найти ключ через 120 дней, то секретное общество из 42 000 членов может найти ключ через 10 дней.

Приведенное выше показывает, что *DES* с размером ключа шифра 56 битов не обеспечивает достаточной безопасности. Позже в этой лекции мы увидим, что есть одно решение этой проблемы – это использование *тройного DES*(*3DES*) с двумя ключами (112 битов) или *тройного DES* с тремя ключами (биты 16).

Слабые ключи. Четыре ключа из 2^{56} *возможных ключей* называются слабыми ключами. Слабые ключи — это одни из тех, которые после операции удаления проверочных бит (используя таблицу 8.12) состоят из всех нулей или всех единиц или половины нулей и половины единиц. Такие ключи показаны в таблице 8.18.

Таблица 8.18. Слабые ключи

Ключи до удаления проверочных бит (64 бита)	Действующие ключи (56 бит)
0101 0101 0101 0101	0000000 0000000
1F1F 1F1F 1F1F 1F1F	0000000 FFFFFFF
E0E0 E0E0 E0E0 E0E0	FFFFFFF 0000000
FEFE FEFE FEFE FEFE	FFFFFFF FFFFFFF

Ключи раунда, созданные от любого из этих слабых ключей, — те же самые и имеют тот же самый тип, что и ключ шифра. Например, эти шестнадцать ключей раунда создают первый ключ, который состоит из всех нулей или всех единиц или наполовину из нулей и единиц.

Это происходит по той причине, что алгоритм генерирования ключей сначала делит ключ шифра на две половины. Смещение или перестановка блока не изменяют блок, если он состоит из всех нулей, или всех единиц, или наполовину из нулей и единиц.

В чем опасность использования слабых ключей? Если мы зашифровали блок слабым ключом и впоследствии расшифровали результат тем же самым слабым ключом, мы получаем первоначальный блок. Процесс создает один и тот же первоначальный блок, если мы расшифровываем блок дважды. Другими словами, каждый слабый ключ есть инверсия самого себя: $E_k(E_k(P)) = P$, как это показано на рис. 8.11.

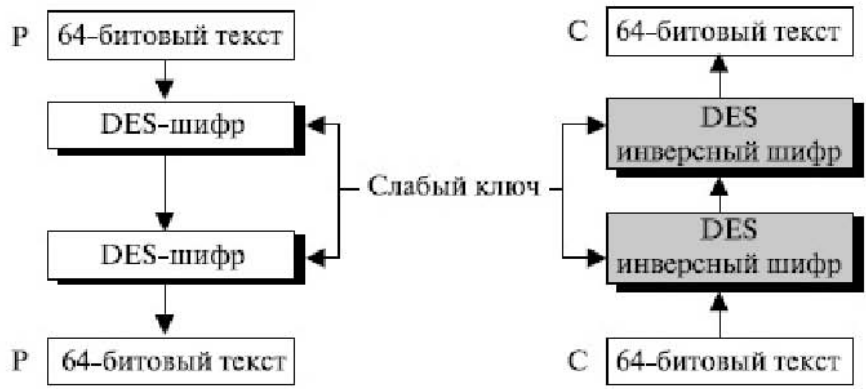


Рис. 8.11. Двойное шифрование и дешифрование со слабым ключом

Слабых ключей надо избегать, потому что противник может легко

распознать их на перехваченном шифре. Если после двух этапов дешифрации результат тот же самый, противник определяет, что он нашел ключ.

Пример 8.8

Давайте попробуем применить первый слабый ключ в таблице 8.18, чтобы два раза зашифровать блок. После того как проведено два шифрования с тем же самым ключом, в результате получим исходный текст. Обратите внимание, что мы ни разу не использовали алгоритм дешифрования, а только провели два раза шифрование.

Ключ: 0x0101 0101 0101 0101
Зашифрованный текст: 0x814FE938589154F7
Исходный текст: 0xl234.56887654321

Ключ: 0x0101 0101 0101 0101
Исходный текст: 0x814FE938589154F7
Зашифрованный текст: 0xl234.56887654321

Полуслабые ключи. Имеются шесть ключевых пар, которые названы полуслабыми ключами. Этим шесть пар показаны в таблице 8.19 (формат на 64 бита перед удалением проверочных бит). Полуслабые ключи создают только два различных ключа раунда и затем повторяют их восемь раз. Кроме того, ключи раунда, созданные от каждой пары, — одни и те же в различном порядке.

Таблица 8.19. Полуслабые ключи

Первый ключ в паре	Второй ключ в паре
01FE 01FE 01FE 01FE	FE01 FE01 FE01 FE01
1FEO 1FEO OEF1 OEF1	E01F E01F F10E F10E
01EO 01E1 01F1 01F1	E001 E001 F101 F101
1FFE 1FFE OEFE OEFE	FE1F FE1F FEOE FEOE
011F 011F 010E 010E	1F01 1F01 OE01 OE01
EOFE EOFE FIFE FIFE	FEEO FEEO FEF1 FEFl

Чтобы проиллюстрировать идею, мы создали ключи раунда от первой пары, как показано ниже:

Ключ раунда 1	9153E54319BD	6EAC1ABCE642
Ключ раунда 2	6EAC1ABCE642	9153E54319BD
Ключ раунда 3	6EAC1ABCE642	9153E54319BD
Ключ раунда 4	6EAC1ABCE642	9153E54319BD
Ключ раунда 5	6EAC1ABCE642	9153E54319BD
Ключ раунда 6	6EAC1ABCE642	9153E54319BD
Ключ раунда 7	6EAC1ABCE642	9153E54319BD
Ключ раунда 8	6EAC1ABCE642	9153E54319BD
Ключ раунда 9	9153E54319BD	6EAC1ABCE642
Ключ раунда 10	9153E54319BD	6EAC1ABCE642
Ключ раунда 11	9153E54319BD	6EAC1ABCE642
Ключ раунда 12	9153E54319BD	6EAC1ABCE642
Ключ раунда 13	9153E54319BD	6EAC1ABCE642
Ключ раунда 14	9153E54319BD	6EAC1ABCE642
Ключ раунда 15	9153E54319BD	6EAC1ABCE642
Ключ раунда 16	6EAC1ABCE642	9153E54319BD

Как показывает список, имеется восемь одинаковых ключей раунда в каждом полуслабом ключе. Кроме того, ключи раунда 1 в первом множестве — те же самые, что и ключи раунда 16 во втором; ключи раунда 2 в первом — те же самые, что и ключи раунда 15 во втором, и так далее. Это означает, что ключи инверсны друг другу: E_{k2}. ($E_{k1}E$(P)) = P, как показано на рис. 8.12.

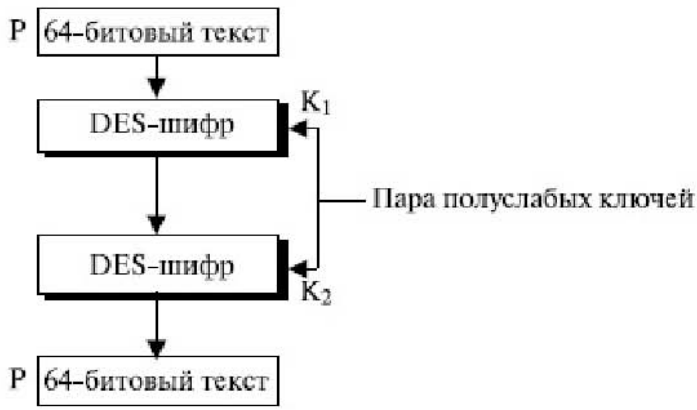


Рис. 8.12. Пара полуслабых ключей при шифровании и дешифровании

Возможно слабые ключи. Также имеется 48 ключей, которые

называются возможно слабыми ключами. Возможно слабый ключ создает только четыре различных ключа раунда; другими словами, шестнадцать ключей раундов разделены на четыре группы, и каждая группа состоит из четырех одинаковых ключей раунда.

Пример 8.9

Какова вероятность случайного выбора слабого, полуслабого или возможно слабого ключа?

Решение

Множество ключей *DES* равно 2^{56}. Общее количество вышеупомянутых ключей — $64 \ (4 + 12 + 48)$. Вероятность выбора одного из этих ключей равна $8,8 \times 10^{-16}$, т.е. исключительно мала.

Ключевое дополнение. Среди множества ключей (2^{56}) некоторые ключи могут быть получены инверсией (изменение из 0 в 1 или 1 в 0) каждого бита в ключе. Ключевое дополнение упрощает процесс *криптоанализа*. Ева может использовать только половину *возможных ключей* (2^{55}), чтобы выполнить атаку грубой силы, потому что

C=E(K,P) -> C=E(K,P)

Другими словами, если мы зашифровали дополнение исходного текста дополнением ключа, мы получаем дополнение зашифрованного текста. Ева не должна проверять все 2^{56} *возможных ключей*, она может проверить только половину из них и затем дополнить результат.

Пример 8.10

Давайте проверим эти сведения о ключах дополнения. Мы используем произвольный ключ и исходный текст, для того чтобы найти соответствующий зашифрованный текст. Если мы имеем ключевое дополнение и исходный текст, то получим i дополнение предыдущего зашифрованного текста (таблица 8.20).

Таблица 8.20. Результаты примера 8.10

	Оригинал	Дополнение
Ключ	1234123412341234	EDCBEDCBEDCBEDCB
Исходный текст	12345678ABCDEF12	EDCBA987543210ED
Зашифрованный текст	E112BE1DEFC7A367	1EED41E210385C98

Кластерный ключ. *Кластерный ключ* рассматривает ситуации, в которых два или более различных ключа создают один и тот же зашифрованный текст из одного и того же исходного текста. Очевидно, каждая пара полуслабых ключей — ключевой кластер. Однако больше кластеров не было найдено. Будущие исследования, возможно, могут открыть некоторые другие.

8.4. Многократное применение DES

Как мы уже видели, основная критика *DES* направлена на *длину ключа*. Возможные технологии и возможности параллельных процессоров делают реальной атаку грубой силы. Одно из решений для улучшения безопасности — это отказ от *DES* и разработка нового шифра. Это решение мы рассмотрим в лекциях 9-10 при применении *AES*. Второе решение — многократное (каскадное) применение множества ключей. Это решение, которое использовалось некоторое время, не требует инвестиций в новое программное обеспечение и аппаратные средства. Ниже рассмотрен такой подход.

Как мы узнали в лекции 7, подстановка, которая размещает все возможные входы во все возможные выходы, является группой с отображениями элементов множества и набором операций. В этом случае использование двух последовательных отображений бесполезно, потому что мы можем всегда найти третье отображение, которое эквивалентно композиции этих двух (свойство замкнутости). Это означает, что если *DES* — группа, то однократный *DES* с ключом k_3 делает то же самое (рисунок 8.13).

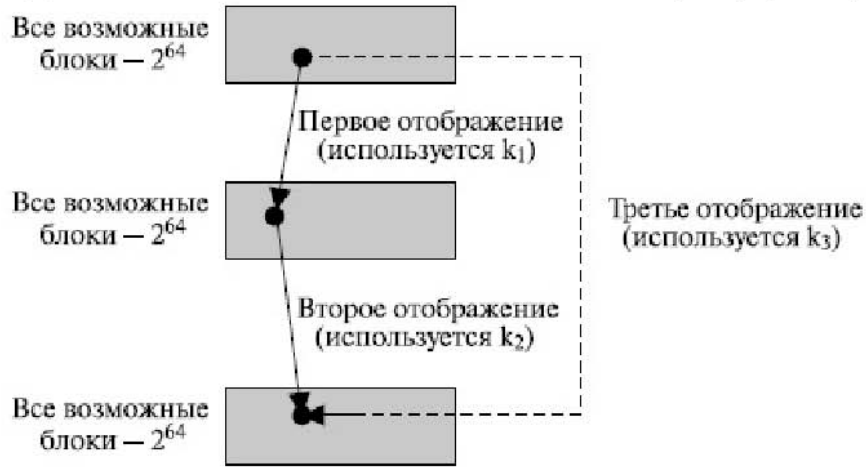


Рис. 8.13. Композиция отображений

К счастью *DES* — не группа. Она базируется на следующих двух параметрах:

а. Номер возможных входов или выходов в *DES* — $N = 2^{64}$. Это означает, что $N! = (2^{64})! = 10^{347\ 380\ 000\ 000\ 000\ 000\ 000}$ отображений. Один из способов способ сделать *DES* группой — это поддержать все эти отображения с размером ключа $\log_2 (2^{64}!) \sim 2^{70}$ битов. Но мы знаем, что длина ключа в *DES* — 56 бит (только маленькая часть этого требуемого огромного ключа).

б. Другой способ сделать *DES* группой – сделать, чтобы множество отображений было подмножеством множества в смысле зависимости от первого параметра; но было доказано, что группы, созданные из группы с помощью первого параметра, имеют ключевой размер 56 битов.

Если *DES* не является группой, то очень маловероятно, что мы можем найти ключ k_3, такой, что

$$E_{k2}(E_{k1}(P)) = E_{k3}(P)$$

Это означает, что мы можем использовать двукратные или трехкратные *DES*, чтобы увеличить размер ключа.

Двукратный DES

Первый подход состоит в том, чтобы использовать двукратный DES (2DES). При этом подходе мы применяем два типа шифров *DES* для шифрования и два типа обратных шифров для дешифрования. Каждый тип использует различный ключ, что означает, что размер ключа теперь удвоился (112 битов). Однако двукратный *DES* уязвим к атаке знания *открытого текста*, как это обсуждается в следующем разделе.

На первый взгляд двукратные *DES* увеличивают число испытаний при поиске ключа от 2^{56} (в однократном *DES*) к 2^{112} (в двукратном *DES*). Однако при использовании атаки знания исходного текста, называемой атакой сведения к середине, можно доказать, что двукратный *DES* улучшает эту устойчивость (до 2^{57} по испытаниям), но не чрезвычайно (к 2^{112}). Рисунок 8.14 показывает диаграмму для двукратного *DES*. Алиса использует два ключа, k_1 и k_2, чтобы зашифровывать исходный текст P в зашифрованный текст C ; Боб использует зашифрованный текст C и два ключа, k_2 и k_1, для восстановления P.

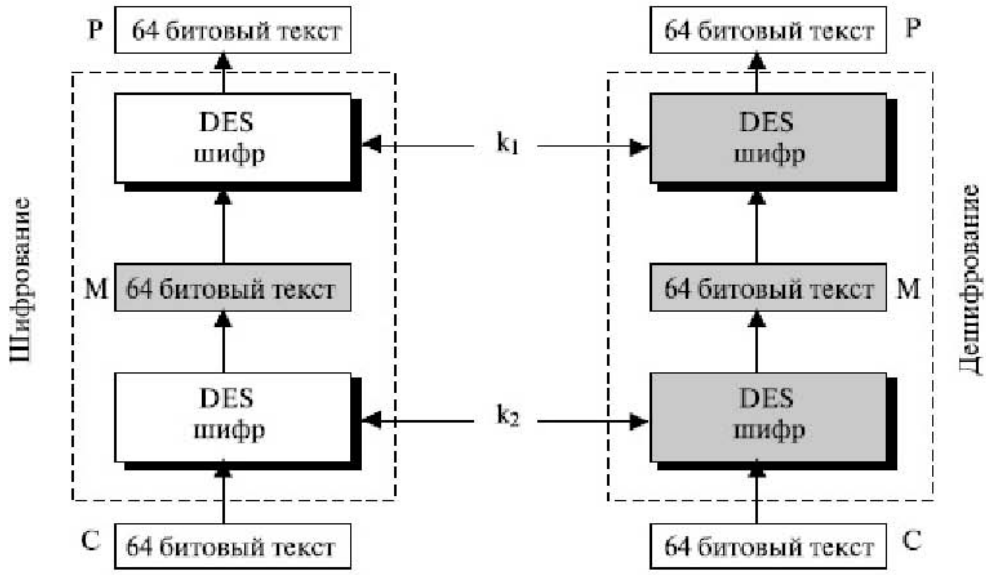


Рис. 8.14. Атака сведения к середине в двукратном DES

В средней точке M — текст, созданный первым шифрованием или первым дешифрованием. Для обеспечения правильной работы он должен быть одинаковым для шифрования и дешифрования. Другими словами, мы имеем два отношения:

$$M = E_{K1}(P) \text{ и } M = E_{K2}(C)$$

Предположим, что Ева перехватила предыдущую пару P и C (атака знания исходного текста). Базируясь на первом отношении из упомянутых выше, Ева зашифровывает P, используя все возможные значения (2^{56}) k_1, и записывает все значения, полученные для M. Базируясь на вторых отношениях, упомянутых выше, Ева расшифровывает C, используя все возможные значения (2^{56}) k_2. Она записывает все значения, полученные для M. Далее Ева создает две таблицы, отсортированные согласно значениям M. Она сравнивает значения для M, пока не находит те пары k_1 и k_2, для которых значение M является одним и тем же в обеих таблицах (как показано на рис. 8.15). Обратите внимание, что должна быть по крайней мере одна пара, потому что она делает исчерпывающий поиск комбинации двух ключей.

1. Если есть только одно соответствие. Ева нашла два ключа (k_1 и k_2). Если есть больше чем один кандидат, Ева перемещается в следующий шаг.

2. Она берет другую перехваченную пару зашифрованного текста и исходного текста и использует каждого кандидата для получения пары ключей, чтобы установить, может ли она получить зашифрованный текст из исходного текста. Если она находит больше чем одного кандидата в виде пары ключей, она повторяет шаг 2, пока, наконец, не находит уникальную пару.

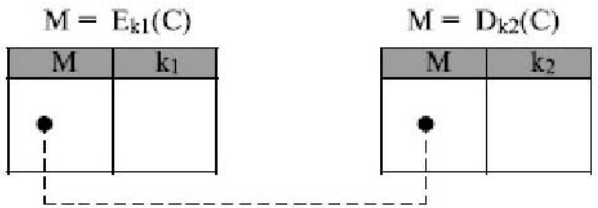


Рис. 8.15. Таблицы для атаки "сведения к середине"

Было доказано, что после применения второго шага к нескольким перехваченным парам "зашифрованный текст — исходный текст" ключи были найдены. Это означает, что вместо того чтобы использовать поиск ключей с помощью 2^{112} испытаний, Ева использует 2^{56} испытаний поиска ключа и проверяет два раза (несколько больше испытаний требуется, если найден на первом шаге единственный кандидат). Другими словами, двигаясь от однократного *DES* до двукратного *DES*, мы увеличили объем испытаний от 2^{56} до 2^{57} (а не до 2^{112}, как это кажется при поверхностном подходе).

Трехкратный DES

Для того чтобы улучшить безопасность *DES*, был предложен трехкратный DES (3DES). Он использует три каскада *DES* для шифрования и дешифрования. Сегодня используются две версии трехкратных *DES*: трехкратный *DES* с двумя ключами и трехкратный *DES* с тремя ключами.

Трехкратный DES с двумя ключами

В трехкратном DES с двумя ключами есть только два ключа: k_1 и k_2. Первый и третий каскады используют k_1 ; второй каскад использует k_2. Чтобы сделать трехкратный *DES* совместимым с *DES*, средний каскад применяет дешифрование (обратный шифр) на стороне шифрования и шифрование (шифр) на стороне дешифрования. Таким способом сообщение, зашифрованное *DES*-ключом k, может быть расшифровано трехкратным *DES*, если $k_1 = k_2 = k$. Хотя трехкратный *DES* с двумя ключами также уязвим при атаке "знания исходного текста", он гораздо устойчивее, чем двукратный *DES*. Он был принят для банков. Рисунок 8.16 показывает трехкратный *DES* с двумя ключами.

Трехкратный DES с тремя ключами

Возможность атак "знания исходного текста" при трехкратном *DES* с двумя ключами "соблазнила" некоторые приложения использовать трехкратный DES с тремя ключами. Алгоритм может применять три каскада шифра *DES* на стороне шифрования и три каскада обратных шифров на стороне дешифрования. Для совместимости с однократным *DES* сторона шифрования использует EDE, а сторона дешифрования — *DED*. E (encryption) — каскад шифрования, D (*decryption*) — каскад дешифрования. Совместимость с однократным *DES* обеспечивается при $k_1 = k$ и установкой k_2 и k_3 к одному и тому же произвольному ключу, выбранному приемником. Трехкратный *DES* с тремя ключами используется многими приложениями, такими как *PGP* (Pretty Good *Privacy*), поскольку гарантирует очень хорошую конфиденциальность.

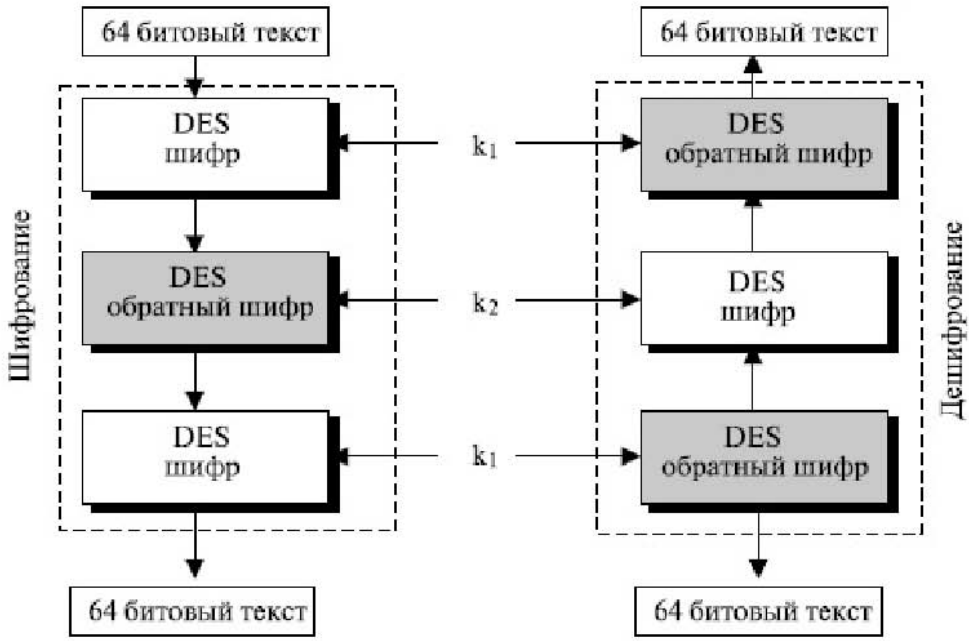


Рис. 8.16. Трехкратный DES с двумя ключами

8.5. Безопасность DES

DES, как первый *блочный шифр*, имеющий важное значение, прошел через много испытаний на безопасность. Среди предпринятых атак лишь три представляют интерес: грубая сила, *дифференциальный криптоанализ* и *линейный криптоанализ*.

Атака грубой силы

Мы уже обсуждали слабость шифра с коротким ключом. Слабость ключа совместно с другими рассмотренными недостатками, делает очевидным, что *DES* может быть взломан с числом испытаний 2^{55}. Однако сегодня большинство приложений использует либо *3DES* с двумя ключами (размер ключа 2^{112}), либо *3DES* с тремя ключами (размер ключа 2^{168}). Эти две многократных версии *DES* позволяют ему показывать существенную стойкость к атакам грубой силы.

Дифференциальный криптоанализ

Мы в лекции 7 уже обсуждали методику *дифференциального криптоанализа* для современных блочных шифров. *DES* не является устойчивым к такому виду атаки. Однако многое указывает, что разработчики *DES* уже знали об этом типе атаки и проектировали S - блоки и специально выбрали *число раундов*, чтобы сделать *DES* стойким к этому типу атаки. Сегодня показано, что *DES* может быть взломан, используя *дифференциальный криптоанализ*, если мы имеем 2^{47} выборок исходного текста или 2^{55} известных исходных текстов. Хотя это выглядит более эффективно, чем в атаке грубой силы, предположить, что кто-то знает 2^{47} выборок исходного текста или 2^{55} выборок исходного текста, практически невозможно. Поэтому мы можем сказать, что *DES* является стойким к дифференциальному *криптоанализу*. Также показано, что увеличение числа раундов до 20 увеличивает число требуемых выборок исходного текста для атаки более чем до 2^{64}. Такое увеличение невозможно, потому что число блоков исходного текста в *DES* только 2^{64}.

Линейный криптоанализ

Мы обсуждали методику *линейного криптоанализа* для современных блочных шифров в лекции 7. *Линейный криптоанализ* — более новая методика, чем *дифференциальный криптоанализ*. *DES* более уязвим к

применению *линейного криптоанализа*, чем к дифференциальному *криптоанализу* — вероятно потому, что этот тип атак не был известен проектировщикам *DES* и S -блоки не являются очень стойкими к линейному *криптоанализу*. Показано, что *DES* может быть взломан с использованием 2^{43} пары известных исходных текстов. Однако с практической точки зрения перехват такого количества пар очень маловероятен.

8.6. Рекомендованная литература

Нижеследующие книги и сайты обеспечивают более детальную информацию о предметах, которые мы обсуждали в этой лекции.

Книги

[Sta06], [Sti06], [Rhe03], [Sal03], [Mao04] и [TW06] — это книги, которые рассматривают *DES*.

Сайты

Нижеследующие сайты содержат боле подробную информацию о темах, обсужденных в этой лекции,

- ссылка: http://www.itl.nist.gov/fipspubs/np46-2.htm - http://www.itl.nist.gov/fipspubs/np46-2.htm
- ссылка: www.nist.gov/director/prog-ofc/report01-2.pdf - http://www.nist.gov/director/prog-ofc/report01-2.pdf
- ссылка: www.engr.mun.ca/-how ard/PAPERS/ldc_tutorial.ps - http://www.engr.mun.ca/-howard/PAPERS/ldc_tutorial.ps
- ссылка: islab.oregonstate.edu/koc/ece575/notes/dc 1.pdf - islab.oregonstate.edu/koc/ece575/notes/dc1.pdf
- ссылка: homes.esat.kuleuven.be / ~ abiryuko/Cryptan/matsui_des - homes.esat.kuleuven.be/~abiryuko/Cryptan/matsui_des

8.7. Итоги

- Стандарт шифрования данных (*DES*) — *блочный шифр* с *симметричными ключами*, изданный национальным Институтом Стандартов и Технологии (*NIST*) как *FIPS* 46 в Федеральном Регистре.
- На стороне шифрования *DES* принимает исходный текст на 64 бита и создает зашифрованный текст на 64 бита. На стороне дешифрования *DES* принимает зашифрованный текст на 64 бита и создает блок на 64 бита исходного текста. Ключ шифра на 56 битов одного типа используется и для шифрования, и для дешифрования.
- Процесс шифрования состоит из двух перестановок (P -блоки), которые называются начальными и конечными перестановками, и шестнадцати раундов Файстеля. Каждый раунд *DES* — шифр Файстеля с двумя элементами (смеситель и устройство замены). Каждый из этих элементов является обратимым.
- Основой *DES* является функция *DES*. Функция *DES* применяет ключ на 48 битов к самым правым 32 битам, чтобы получить на выходе 32 бита. Эта функция составлена из четырех операций: перестановки расширения, отбеливателя (который добавляет ключ), группы S -блоков и прямой перестановки.
- Генератор ключей раунда создает из ключа шифра на 56 битов шестнадцать ключей по 48 битов. Однако ключ шифра обычно представляется как ключ на 64 бита, в котором 8 дополнительных битов являются проверочными битами — они отбрасываются перед фактическим процессом генерирования ключей.
- *DES* показывает хорошие рабочие характеристики по отношению к эффектам полноты (законченности) и лавины. К числу слабостей *DES* относятся: построение шифра (S -блоки и P -блоки) и ключ шифра (длина), слабые ключи, полуслабые ключи, возможно слабые ключи и дополнение ключей.
- Так как *DES* — не группа, одно из решений по улучшению безопасности *DES* состоит в том, чтобы пользоваться многократными *DES*, которые используют кратное число применения ключей (двукратный или трехкратный *DES*). Двукратный *DES* уязвим к атаке сведения к середине, поэтому в обычных приложениях используется трехкратный *DES* с двумя ключами или с тремя ключами.
- Разработка S -блоков и *число раундов* сделали *DES* почти

обладающим иммунитетом от *дифференциального криптоанализа*. Однако *DES* уязвим при применении *линейного криптоанализа*, если злоумышленники смогут собрать достаточно много исходных текстов.

8.8. Набор для практики

Обзорные вопросы

1. Каков размер блока в *DES*? Каков размер ключа шифра в *DES*? Каков размер ключей раунда в *DES*?
2. Каково *число раундов* в *DES*?
3. Сколько смесителей и устройств замены используется в первом способе шифрования и обратного дешифрования? Сколько их используется при втором способе?
4. Сколько перестановок используется в алгоритме шифра *DES*?
5. Сколько операций *ИСКЛЮЧАЮШЕЕ ИЛИ* используется в *DES*-шифре?
6. Почему *DES*-функции необходима расширяющая перестановка?
7. Почему генератор ключей раунда нуждается в удалении проверочных бит?
8. Какова разность между слабым ключом, полуслабым ключом и возможно слабым ключом?
9. Что такое двукратный *DES*? Какая атака двукратного *DES* сделала его бесполезным?
10. Что такое трехкратный *DES*? Что такое трехкратный *DES* с двумя ключами? Что такое трехкратный *DES* с тремя ключами?

Упражнения

1. Ответьте на следующие вопросы об `S`-блоках в *DES*:
   - покажите результат прохождения `110111` через `S`-блок `3`.
   - покажите результат прохождения `001100` через `S`-блок `4`.
   - покажите результат прохождения `000000` через `S`-блок `7`.
   - покажите результат прохождения `111111` через `S`-блок `2`.
2. Нарисуйте таблицу, которая показывает результат прохождения

000000 через все 8 S -блоков. Рассмотрите результат на выходе.

3. Нарисуйте таблицу, которая показывает результат прохождения 111111 через все 8 S -блоков. Рассмотрите результат на выходе.
4. Проверьте третий критерий для S -блока 3, используя следующие пары входов:
   - 000000 и 000001
   - 111111 и 111011
5. Проверьте четвертый критерий построения S -блока 2, используя следующие пары входа:
   - 001100 и 110000
   - 110011 и 001 111
6. Проверьте пятый критерий построения S -блока 4, используя следующие пары входов:
   - 001100 и 110000
   - 110011 и 001 111
7. Создайте 32 6 -битовые входные пары, чтобы проверить шестой критерий построения S -блока 5.
8. Покажите, как выполнены восемь критериев построения S -блока 7.
9. Докажите первый критерий построения P -блоков, проверив входы к S -блоку 2 раунда 2.
10. Докажите второй критерий построения для P -блоков, проверяя входы к S -блоку раунда 4.
11. Докажите третий критерий построения P -блоков, проверяя выход S -блока раунда 3.
12. Докажите четвертый критерий построения P -блоков, проверяя выход S -блока 6 раунда 12.
13. Докажите пятый критерий проекта для P -блоков, проверяя отношение между S -блоками 3, 4 и 5 в раундах 10 и 11.
14. Докажите шестой критерий построения P -блоков, проверяя отношение S -блока конечного пункта произвольного блока.
15. Докажите седьмой критерий проекта для P -блоков, проверяя отношения между S -блоком 5 в раунде 4 и S -блоком 7 в раунде 5.
16. Измените рисунок 8.9, используя альтернативный подход.
17. Докажите, что обратный шифр на рис. 8.9 — фактически инверсия шифра *DES* с тремя раундами. Стартуя с исходного текста и

начального шифра, докажите, что вы можете получить тот же самый исходный текст в конце процесса применения обратного шифра.

18. Тщательно изучите ключ при перестановке сжатия таблицы 8.14.
    - Какие входные порты отсутствуют на выходе?
    - Все ли левые 24 бита выхода идут от всех левых 28 входных бит?
    - Все ли правые 24 бита выхода идут от всех 28 входных битов?

19. Покажите результат следующих шестнадцатеричных данных

    0110 10234110 1023

    после прохождения их через начальный блок перестановки.

20. Покажите результат следующих шестнадцатеричных данных

    AAAA BBBB CCCC DDDD

    после прохождения их через конечный блок перестановки.

21. Если ключ с проверочными битами (64 бита) — 0123 *ABCD* 2562 1456, найдите ключ первого раунда.
22. Используя блок исходного текста всех нулей и ключ на 56 битов из всех нулей, докажите слабость ключевого дополнения, предполагающую, что *DES* состоит только из одного раунда.
23. Вы можете изобрести атаку "сведение к середине" для трехкратного *DES*?
24. Напишите *псевдокод* для переставляющей процедуры, используемой в алгоритме 8.1:

    permute (n,m, inBlock [n], outBlock [m], permutationTable [m])

25. Напишите *псевдокод* для процедуры разбиения, используемой в алгоритме 8.1:

    spilt(n, m, inBlock [n], leftBlock, rightBlock [m])

26. Напишите *псевдокод* для процедуры объединения, используемой в

алгоритме 8.1:

combine (n, m, leftBlock [n], rightBlock [n1, outBlock [m])

27. Напишите *псевдокод* для процедуры *ИСКЛЮЧАЮШЕЕ ИЛИ*, используемой в алгоритме 8.1:

exclusiveOr (n, firstInBlock [n], второй! nBlock [n], outBlock [n])

28. Измените алгоритм 8.1, чтобы представить альтернативный подход.
29. Дополните алгоритм 8.1, чтобы использовать его и для шифрования, и для дешифрования.

Преобразования

В этой лекции мы обсуждаем Усовершенствованный стандарт шифрования (AES — ADVANCED ENCRYPTION STANDARD) — современный блочный шифр с симметричными ключами, который может заменить DES.

9.1. Введение

Усовершенствованный стандарт шифрования (ADVANCED ENCRYPTION STANDARD) — стандарт на *блочный шифр* с *симметричными ключами*, изданный Национальным Институтом Стандартов и Технологии (*NIST*) в декабре 2001 года.

История

В 1997 году *NIST* начал искать замену для *DES*, которая была названа Усовершенствованным стандартом шифрования (ADVANCED ENCRYPTION STANDARD или *AES*). В *NIST*-спецификациях были заложены требования размера блока из 128 битов и трех различных размеров ключей: 128, 192 и 256 битов. Спецификации также требовали, чтобы *AES* был открытым алгоритмом, публично доступным во всем мире.

Спецификации стандартов были объявлены по многим странам, чтобы запросить ответы у специалистов и заинтересованных лиц со всех континентов.

После Первой Конференции по выбору Кандидатов AES *NIST* объявила, что 15 из 21 полученных алгоритмов отвечают поставленным требованиям и выбраны как первые кандидаты (август 1998 г.). Алгоритмы были представлены от многих стран; разнообразие этих предложений демонстрировало открытость процесса и участие всего мира.

После Второй Конференции по выбору Кандидата AES, которая была проведена в Риме, *NIST* объявила 5 из 15 из кандидатов. Алгоритмы

MARS, *RC6*, *Rijndael*, *Serpent* и *Twofish* были выбраны как финалисты (август 1999).

После Третьей Конференции по выбору Кандидата AES *NIST* объявила, что выбран алгоритм Усовершенствованного Стандарта Шифрования — Rijndael, спроектированный бельгийскими исследователями Джоном Даеменом и Винсентом Риджменом (октябрь 2000 г.).

В феврале 2001 г. *NIST* объявил, что эскиз Федерального Стандарта обработки Информации (FIPS) доступен для общественного рассмотрения и комментариев.

Наконец, *AES* был издан как *FIPS* 197 в Федеральном Регистре в декабре 2001 г.

Критерии

Критерии, определенные *NIST* для выбора *AES*, относятся к трем областям: безопасность, стоимость и реализация. В конце концов Rijndael был оценен в совокупности как лучший, отвечающий этим критериям.

Безопасность

Особое внимание уделялось безопасности. Поскольку *NIST* ясно потребовал `128`-битовый ключ, то этот критерий определял то, что внимание обращалось на устойчивость шифра к другим атакам *криптоанализа*, нежели атака грубой силы.

Стоимость

Вторым критерием была стоимость, которая задает требуемую вычислительную эффективность и требования для различных реализаций, таких как аппаратные средства, программное обеспечение или *интеллектуальные карты* доступа.

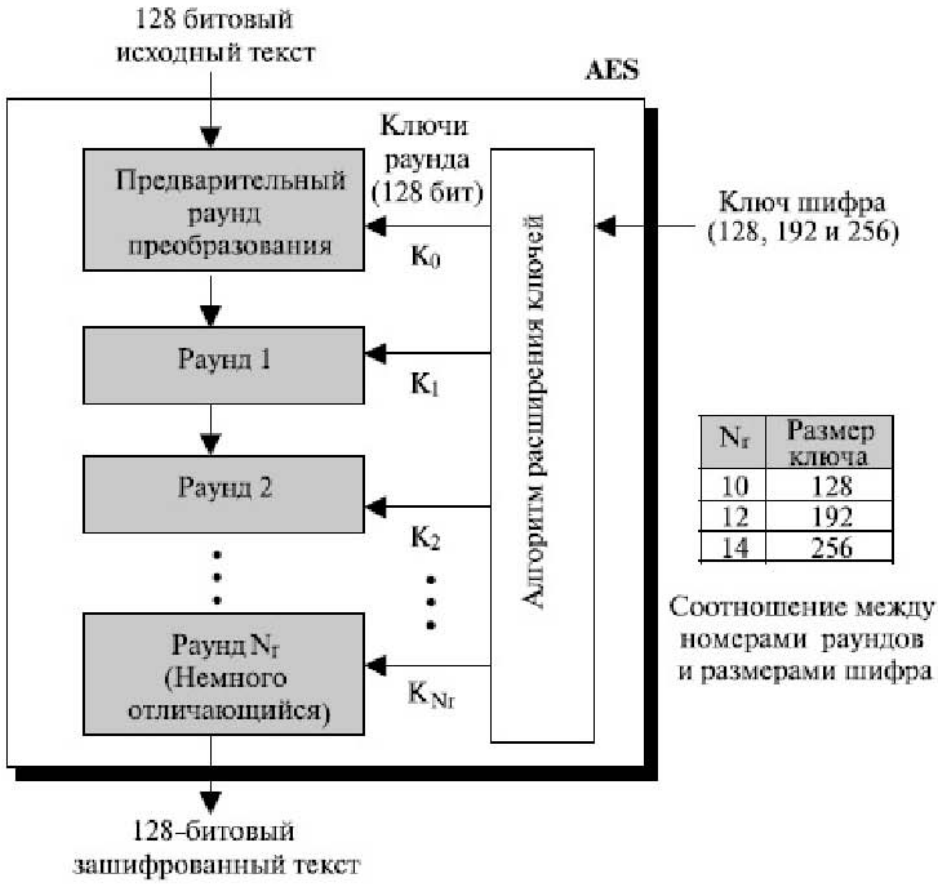
Этот критерий включал требование, что алгоритм должен иметь гибкость (возможность быть реализованным на любой платформе) и простоту.

Раунды

AES — шифр не-Файстеля, который зашифровывает и расшифровывает блок данных `128` битов, используя `10`, `12` или `14` раундов. Размер ключа может быть `128`, `192` или `256` битов и зависит от номера раунда. Рисунок 9.1 показывает общую схему: алгоритм шифрования (называемого шифром); алгоритм дешифрования (называемый обратным шифром), для которого применяются те же ключи, но в обратном порядке.

На рис. 9.1 N определяет номер раундов. Рисунок также показывает отношение между номером раунда и размером ключа — это означает, что мы имеем три различных версии *AES*; они обозначаются как *AES*-128, *AES*-192 и *AES*-256. Однако ключи раунда, которые созданы алгоритмом расширения ключей всегда `128` бит, имеют тот же самый размер, что и блоки зашифрованного или исходного текста.

AES определил три версии, с 10, 12 и 14 раундами. Каждая версия использует различный размер ключа шифра (128, 192 или 256), но ключ раунда — всегда 128 бит.



N_r	Размер ключа
10	128
12	192
14	256

Соотношение между номерами раундов и размерами шифра

Рис. 9.1. Общее построение шифрования AES шифра

Число ключей раунда, сгенерированных алгоритмом расширения ключей, всегда на один больше, чем *число раундов*. Другими словами, мы имеем

номер ключей раунда = $N_r + 1$

Мы обозначаем ключи раунда как K_0, K_1, $K_2 \ldots$, K_N.

Единицы данных

AES использует пять единиц для представления данных: биты, байты,

слова, блоки и массивы состояний. Бит — наименьшая и элементарная единица; другие единицы могут быть выражены в терминах меньших единиц Рисунок 9.2 показывает единицы неэлементарных данных: байт, слово, блок, массив состояний (state).

Бит

В *AES* бит — двоичная цифра со значением `0` или `1`. Мы используем строчные буквы для обозначения бит.

Байт

Байт — группа из восьми битов, которая может быть обработана как единый объект: матрица из одной строки (1×8) восьми битов или столбец матрицы (8×1) из восьми битов. Когда информация байта обрабатываются как матрица строки, то биты вставляются в матрице слева направо. Когда байт обрабатывается как матрица столбца, биты вставляются в матрице сверху вниз. Мы будем использовать строчную "жирную" букву (Bold) для обозначения байта.

Слово

Слово — группа из `32` битов, которая может быть обработана как единый объект. Это матрица из строки в четыре байта или столбец матрицы из четырех байтов. Когда слово обрабатывается как матрица-строка, байты вставляются слева направо. Когда слово представляется матрицей-колонкой, байты вставляются сверху вниз. Мы будем использовать строчную "жирную" букву `W` для обозначения слова.

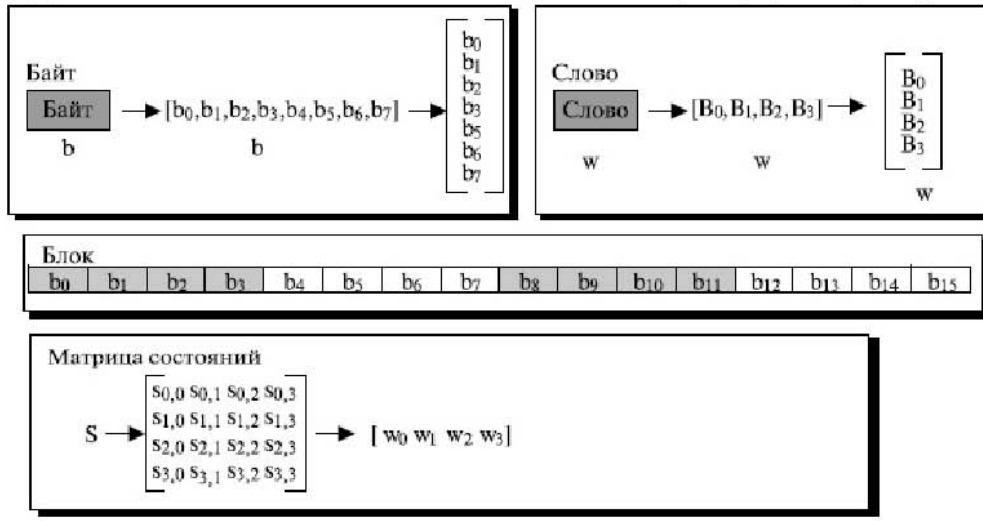


Рис. 9.2. Единицы данных, используемых в AES

Блок

AES зашифровывает и расшифровывает блоки данных. Блок в *AES* — группа `128` битов. Однако блок может быть представлен как матрица-строка из `16`-ти байтов.

Матрица состояний

AES использует несколько раундов, каждый раунд состоит из несколько каскадов. Блок данных преобразовывается от одного каскада к другому. В начале и в конце шифра *AES* применяется термин блок данных; до и после каждого каскада блок данных называется матрицей состояний. Мы используем "жирную" заглавную букву, чтобы обозначить эту матрицу. Хотя матрица состояний на различных каскадах обычно обозначается `S`, мы иногда применяем букву `T`, чтобы обозначить временную матрицу состояний. Матрицы состояний, подобно блокам, состоят из `16` байтов, но обычно обрабатываются как матрицы 4×4 байтов. В этом случае каждый элемент матрицы состояний обозначается как $S_{r,c}$, где `r` (от `0` до `3`) определяет строку и `c` (от `0` до `3`) определяет столбец. Иногда матрица состояний обрабатывается как

матрица-строка слов (1×4). Это имеет смысл, если мы представляем слово как матрицу-столбец. В начале шифра байты в блоке данных вставляются в матрицу состояний столбец за столбцом, в каждом столбце — сверху вниз. В конце шифра байты в матрице состояний извлекаются, как это показано на рис. 9.3.

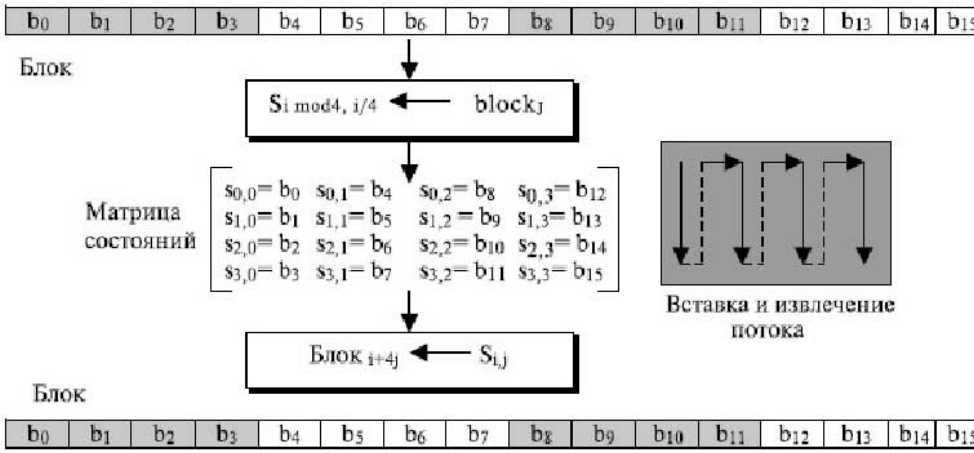


Рис. 9.3. Преобразование блок – матрица состояний и матрица состояний в блок

Пример 9.1

Рассмотрим, как можно изобразить блок с 16 символами в виде матрицы 4×4. Предположим, что текстовый блок — "*AES* uses a matrix". Добавим два фиктивных символа в конце и получим "AESUSESAMATRIXZZ". Теперь мы заменим каждый символ целым числом между 00 и 25. Представим каждый байт как целое число с двумя шестнадцатеричными цифрами. Например, символ "S" сначала поменяем на 18, а затем запишем в шестнадцатеричном изображении как 12. Матрица состояний тогда заполняется столбец за столбцом, как это показано на рис. 9.4.

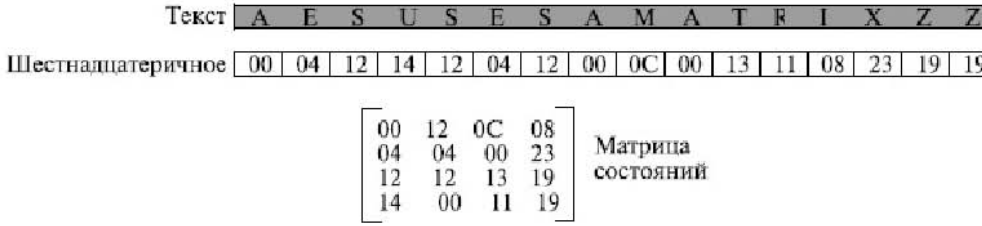


Рис. 9.4. Переход шифрованного текста в матрицу состояний

Структура каждого раунда

Рисунок 9.5 показывает структуру каждого раунда на стороне шифрования. Каждый раунд, кроме последнего, использует четыре преобразования, которые являются обратимыми. Последний раунд имеет только три преобразования.

Как показывает рис. 9.5, каждое преобразование принимает матрицу состояний и создает другую матрицу состояний, которая применяется для следующего преобразования или следующего раунда. Секция, предваряющая раунд, использует только одно преобразование (*AddRoundKey*); последний раунд использует только три преобразования (`MixColumns` — преобразование отсутствует).

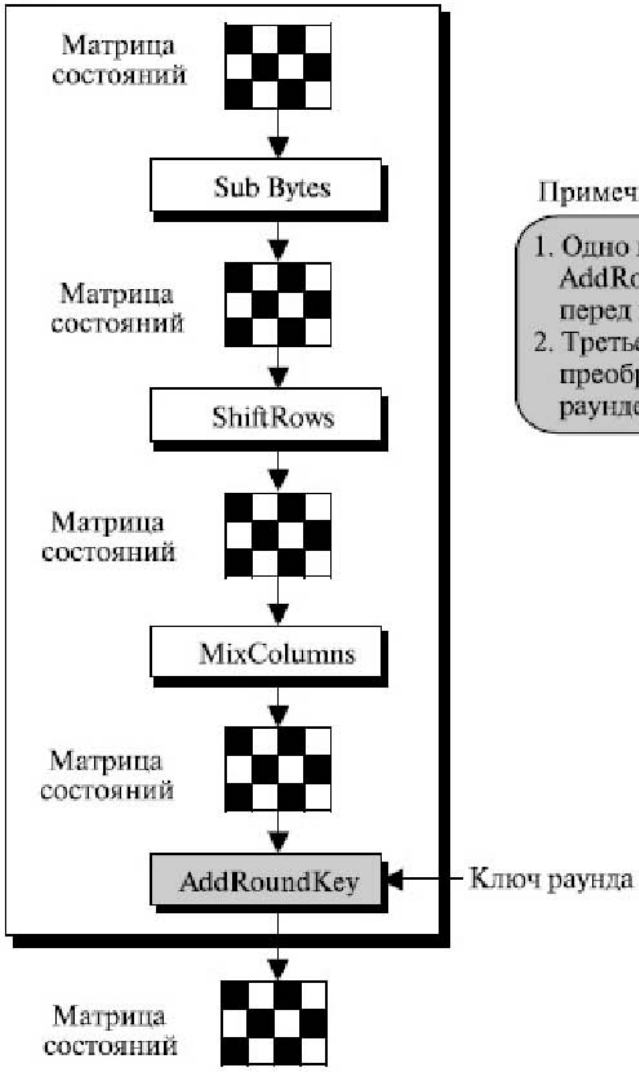


Рис. 9.5. Структура каждого раунда на стороне шифрования

9.2. Преобразования

Чтобы обеспечить безопасность, *AES* использует четыре типа преобразований: подстановка, перестановка, смешивание и добавление ключа. Ниже мы обсудим каждое.

Подстановка

AES подобно *DES* использует подстановку. Однако этот механизм имеет различия. Первое: подстановка делается для каждого байта. Второе: для преобразования каждого байта используется только одна таблица — это означает, что если два байта одинаковы, то и результат преобразования одинаков. Третье: преобразование определяется или процессом поиска в таблице, или математическим вычислением в $GF(2^8)$ поле. *AES* работает с двумя обратимыми преобразованиями.

SubBytes

Первое преобразование, SubBytes, используется на стороне шифрования. Чтобы применить подстановку к байту, мы интерпретируем байт как две шестнадцатеричные цифры. Левая цифра определяет строку, а правая — колонку в таблице перестановки. На пересечении строки и колонки, обозначенных этими шестнадцатеричными цифрами, находится новый байт. Рисунок 9.6 иллюстрирует идею.

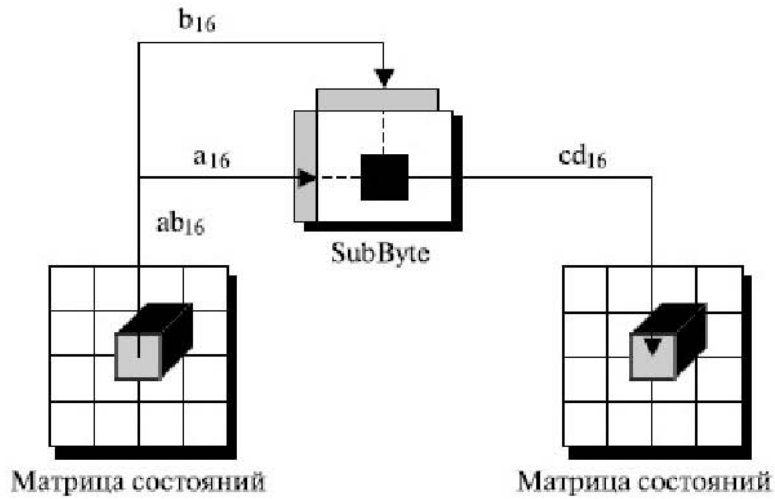


Рис. 9.6. Преобразование SubByte

В преобразовании *SubBytes* состояние обрабатывается как матрица байтов 4×4. В один момент проводится преобразование одного байта. Содержание каждого байта изменяется, но расположение байтов в матрице остается тем же самым. В процессе преобразования каждый

байт преобразуется независимо от других — это шестнадцать личных преобразований байт в байт.

Операция SubByte включает 16 независимых преобразований байта в байт.

Таблица 9.1 показывает таблицу подстановки (S-блок) для преобразования *SubBytes*. Преобразование обеспечивает эффект перемешивания. Например, два байта, $5A_{16}$ и $5B_{16}$, которые отличаются только одним битом (самый правый бит), преобразованы в BE_{16} и 39_{16}, которые отличаются четырьмя битами.

Таблица 9.1. Таблица преобразования *SubBytes*

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	*FE*	D7	AB	76
1	CA	82	C9	7D	FA	59	47	FO	AD	D4	A2	AF	9C	A4	72	CO
2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
4	09	83	2C	1A	1B	6E	*5A*	AO	52	3B	D6	B3	29	E3	2F	84
5	53	D1	00	ED	20	*FC*	B1	5B	6A	CB	BE	39	4A	4C	58	CF
6	DO	EF	AA	FB	43	4D	33	85	45	F9	02	F7	50	3C	9F	A8
7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DE
A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
B	E7	CB	37	6D	8D	D5	4E	A9	6C	56	F4	*EA*	65	7A	AE	08
C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	IF	4B	BD	8B	8A
D	70	3E	B5	66	48	03	F6	OE	61	35	57	B9	86	C1	1D	9E
E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
F	8C	A1	89	OD	BF	E6	42	68	41	99	2D	OF	BO	54	BB	16

InvSubBytes

InvSubBytes — *инверсия* SubBytes. Преобразование сделано с использованием таблицы 9.2, и мы можем легко проверить, что эти два преобразования являются обратными друг другу.

Таблица 9.2. Таблица преобразования InvSubBytes

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	52	09	6A	D5	30	36	A5	38	BF	40	A3	9E	81	F3	D7	FB
1	7C	E3	39	82	9B	2F	FF	87	34	8E	43	44	C4	DE	E9	CB
2	54	7B	94	32	A6	C2	23	3D	EE	4C	95	0B	42	FA	C3	4E
3	08	2E	A1	66	28	D9	24	B2	76	5B	A2	49	6D	8B	D1	25
4	72	F8	F6	64	86	68	98	16	D4	A4	5C	CC	5D	65	B6	92
5	6C	70	48	50	FD	ED	B9	DA	5E	15	46	57	A7	8D	9D	84
6	90	D8	AB	00	8C	BC	D3	0A	F7	E4	58	05	B8	B3	45	06
7	D0	2C	1E	8F	CA	3F	0F	02	C1	AF	BD	03	01	13	8A	6B
8	3A	91	11	41	4F	67	DC	*EA*	97	F2	CF	CE	F0	B4	E6	73
9	96	AC	74	22	E7	AD	35	85	E2	F9	37	E8	1C	75	DE	6E
A	47	E1	1A	71	1D	29	C5	89	6F	B7	62	0E	AA	18	BE	1B
B	*FC*	56	3E	4B	C6	D2	79	20	9A	*DB*	C0	*FE*	78	CD	*5A*	F4
C	1F	D	A8	33	88	07	C7	31	B1	12	10	59	27	80	EC	5F
D	60	51	7E	A9	19	B5	4A	0D	2D	E5	7A	9F	93	C9	9C	EF
E	A0	E0	3B	4D	AE	2A	F5	B0	CB	EB	BB	3C	83	53	99	61
F	17	2B	04	7E	BA	77	D6	26	E1	69	14	63	55	21	0C	7D

Пример 9.2

Рисунок 9.7 показывает, как матрица состояний преобразуется с использованием `SubBytes`. Рисунок также показывает, что `InvSubBytes` однозначно воспроизводит оригинал. Заметим, что если два байта имеют одинаковое значение, то они преобразуются одинаково. Например, два байта 04_{16} и 04_{16} в левой матрице состояний преобразуются в $F2_{16}$ и $F2_{16}$ в правой матрице состояний и наоборот. Причина в том, что каждый байт использует одну и ту же таблицу преобразований.

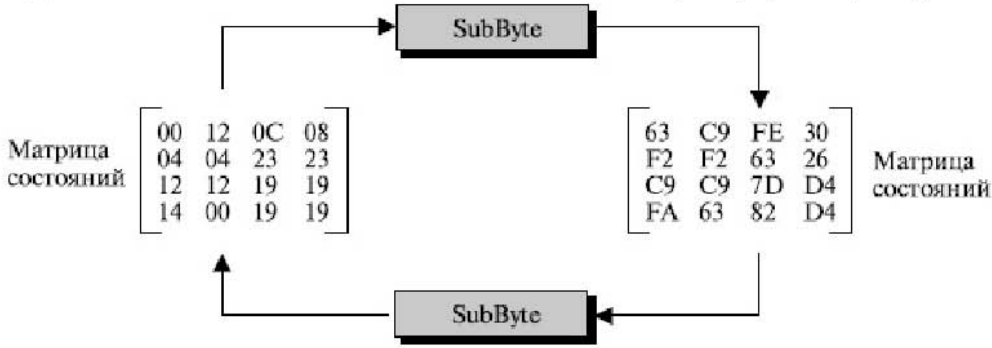


Рис. 9.7. Преобразование SubByte по примеру 9.2.

Преобразование с использованием поля GF(2 в степени 8)

Хотя мы можем использовать таблицы 9.1 и 9.2 для перестановки каждого байта, *AES* дает определение алгебраическим преобразованиям на основе поля $GF(2^8)$ с помощью неприводимых полиномов $(x^8 + x^4 + x^3 + x + 1)$, как это показано на рис. 9.8.

Преобразование *SubByte* повторяет процесс, называемый *subbyte*, шестнадцать раз. Inv *SubByte* повторяет процесс, называемый invsubbyte. Каждый шаг преобразования обрабатывает один байт.

В процедуре *subbyte* байт (двоичной строки на 8 битов) находится в $GF(2^8)$ с помощью неприводимого полинома по модулю $(x^8 + x^4 + x^3 + x + 1)$. Обратите внимание, что байт — 00_{16} сам является собственной *инверсией*. *Инвертированный* байт затем интерпретируется как матрица-столбец с самым младшим битом наверху и самым старшим битом внизу. Эта матрица-столбец умножается на постоянную квадратную матрицу, X, и результат, который является матрицей-столбцом, складывается с постоянной матрицей столбца y, что дает новый байт. Обратите внимание, что умножение и сложение битов происходит в GF(2). invsubbyte делает те же самые действия в обратном порядке.

После нахождения байта инверсного сомножителя процесс похож на аффинное шифрование, которое мы обсуждали в лекции 4. При

шифровании умножение является первой операцией, сложение — второй. При дешифровании вычитание (сложение *инверсией*) является первым, а деление (умножение с *инверсией*) — вторым. Мы можем легко доказать, что эти два преобразования инверсны друг другу, потому что сложение или вычитание в GF(2) — фактически операция *ИСКЛЮЧАЮШЕЕ ИЛИ*.

subbyte:

$$d = X\ (s_{r,c})^{-1} \oplus y$$

invsubbyte:

$$[X^{-1}(d \oplus y)^{-1}] = [X^{-1}(X((s_{r,c})^{-1}y \oplus y]^{-1} = [(s_{r,c})^{-1}]^{-1} = s_{r,c}$$

Преобразования SubBytes и InvSubBytes инверсны друг другу.

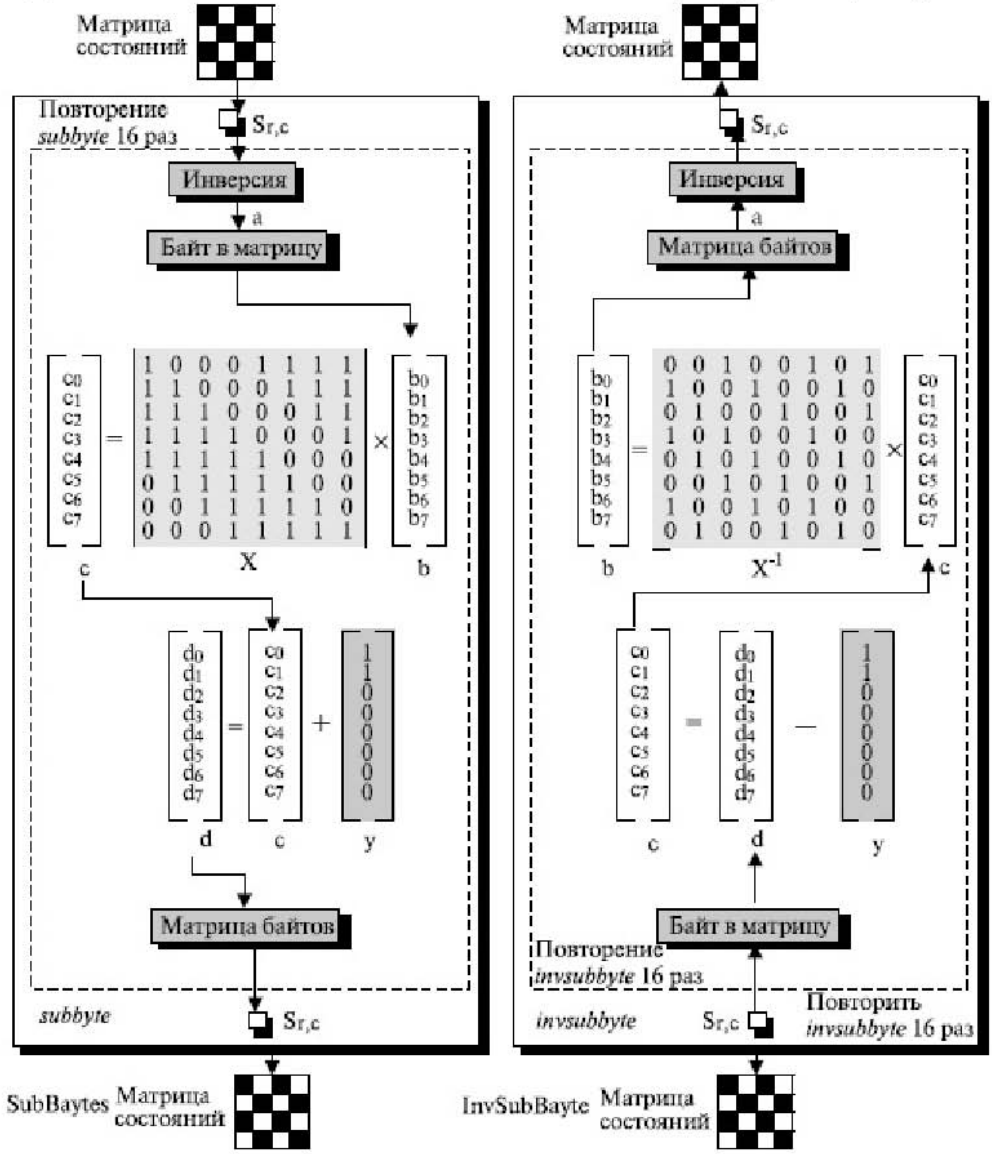


Рис. 9.8. Процессы Subbytes и Invbytes

Пример 9.3

Покажем, как байт `0C` преобразуется в *`FE`* с помощью процедуры *`subbyte`* и преобразуется обратно в `0C` с помощью процесса `invsubbyte`.

1. *subbyte*

a. Инверсный сомножитель в поле GF(2^8) есть B0 или в двоичном отображении b (1011 0000).

b. Умножая на матрицу X эту матрицу, имеем в результате c = (10011101).

c. Результат применения операции *XOR* (*ИСКЛЮЧАЮШЕЕ ИЛИ*) - d = (11111110) *FE* в шестнадцатеричном представлении.

2. invsubbyte

a. Результат применения операции *XOR* (*ИСКЛЮЧАЮШЕЕ ИЛИ*) - c = (10011101)

b. Результат умножения на матрицу X^{-1} - (11010000) или B0.

c. *Инверсия* по умножению B0 - это 0C.

Алгоритм

Хотя на рисунке мы показали матрицы, чтобы подчеркнуть характер подстановки (аффинное преобразование), алгоритм не обязательно использует умножение и сложение матриц, потому что большинство элементов в постоянной квадратной матрице — только 0 или 1. Значение постоянной матрицы столбца — 0×63. Мы можем написать простой алгоритм, чтобы выполнить *SubByte*. Алгоритм 9.1 вызывает процедуру *subbyte* 16 раз — один раз для каждого байта в матрице состояний. Процедура ByteToMatrix преобразовывает байт к матрицу-столбец 8×1. Процедура MatrixToByte преобразовывает матрицу-столбец 8×1 к байту. Расширение этого алгоритма для InvSubBytes оставляем как упражнение.

Нелинейность

Хотя умножение и сложение матриц в процедуре *subbyte* —

преобразование аффинного типа и линейно, замена байта его мультипликативной *инверсией* в $GF(2^8)$ — нелинейная. Этот шаг делает все преобразование нелинейным.

Перестановка

Другое преобразование производит сдвиг в раунде. Этот сдвиг переставляет байты. В отличие от *DES*, в котором делается поразрядная перестановка, преобразование сдвига делается на уровне байта; порядок битов в байте не меняется.

```
SubBytes (S)
{
for (r = 0 to 3)
   for (c = 0 to 3)
   S_{r,c} = subbyte(S_{r,c})
}

subbyte (byte)
{
  a ← byte^{-1}     //Multiplicative inverse in GF(2^8) with inverse
        // of 00 to be 00
  ByteToMatrix (a,b)
  For (i= 0 to 7)
  {
    c_i ← b_i ⊕ b_{(i+4) mod 8} ⊕ b_{(i+5) mod 8} ⊕ b_{(i+6) mod 8} ⊕ b_{(i+7) mod 8}
    d_i ← c_i ⊕ ByteToMatrix(0 × 63)
   }
  MatrixToByte (d,d)
  byte ← d
}
```

Пример 9.1. Программа на псевдокоде для преобразования SubBytes

ShiftRows

При шифровании применяется преобразование, называемое ShiftRows, со смещением влево. Число сдвигов зависит от номера строки (`0, 1, 2` или `3`) матрицы состояний. Это означает, что строка `0` не сдвигается и последняя строка сдвигается на три байта. Рисунок 9.9 показывает преобразование смещения.

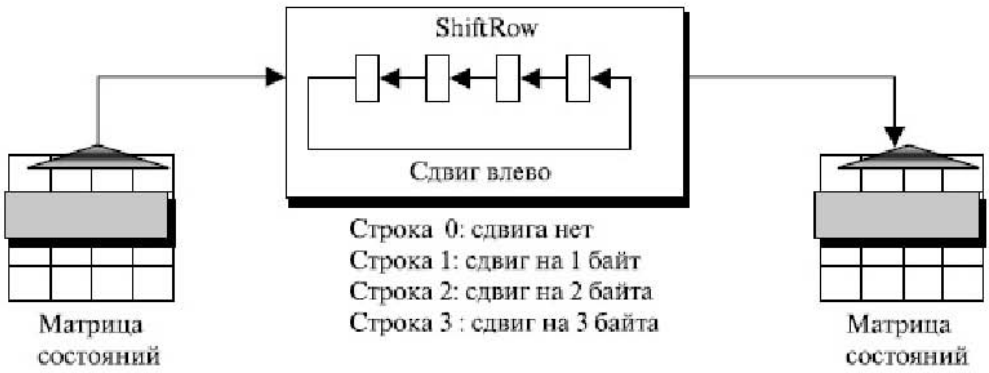


Рис. 9.9. Преобразование ShiftRows

Обратите внимание, что преобразование `ShiftRows` работает одновременно только с одной строкой.

InvShiftRows

При дешифровании применяется преобразование, называемое InvShiftRows, со смещением вправо. Число сдвигов равно номеру строки (`0, 1, 2` и `3`) в матрицы состояний.

ShiftRows- и InvShiftRows- преобразования инверсны друг другу.

Алгоритм

Алгоритм 9.2 для преобразования `ShiftRows` очень прост. Однако чтобы подчеркнуть, что преобразование делается одновременно только с одной строкой, мы используем процедуру, называемую `shiftrow`,

которая сдвигает байт в единственной строке. Мы вызываем эту процедуру три раза.

Процедура `shiftrow` сначала копирует строку во временную матрицу строки (матрица `t`), а потом сдвигает строку.

Пример 9.4

Рисунок 9.10 показывает, как, используя преобразование `ShiftRows`, преобразуется матрица состояний. Рисунок также иллюстрирует, как преобразование `InvShiftRows` создает первоначальную матрицу состояний.

```
{
  for (r = 1 to 3)
  shiftrow (S_r, r)      // s_r r-тая строка}

  shiftrow (row, n)  // n - число байтов, на которое должен
          // быть сделан сдвиг

  {
   CopyRow (row, t) for (c = 0 to 3) // t - временная строка
   for (c = 0 to 3)
        row(c−n) mod 4 ← t_c
  }
```

Пример 9.2. Программа на псевдокоде для преобразования ShiftRows

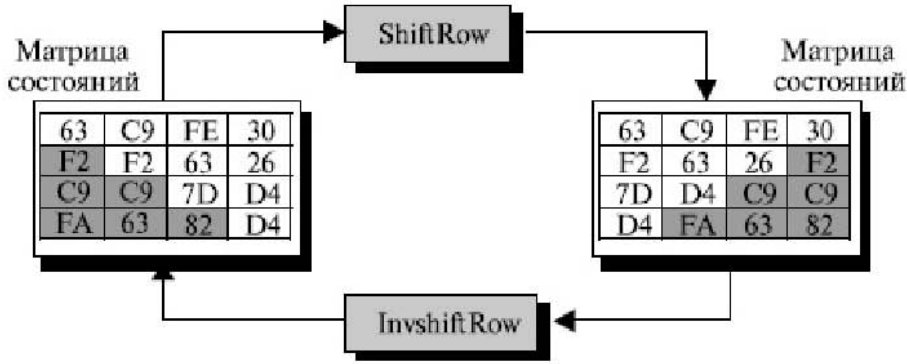


Рис. 9.10. Пример преобразования ShiftRow в примере 9.4.

Смешивание

Подстановка, которая делается преобразованием `SubByte`, изменяет значение байта, основанное только на первоначальном значении и входе в *таблице; процесс* не включает соседние байты. Мы можем сказать, что `SubByte` — внутрибайтовое преобразование. Перестановка, которая делается `ShiftRows`-преобразованием, обменивает местами байты, не переставляя биты в байтах. Мы можем сказать, что `ShiftRows` — преобразование обмена байтами. Теперь нам нужно внутрибайтовое преобразование, изменяющее биты в байте и основанное на битах в соседних байтах. Мы должны смешать байты, чтобы обеспечить рассеивание на разрядном уровне.

Преобразование смешивания изменяет содержание каждого байта, преобразовывая четыре байта одновременно и объединяя их, чтобы получить четыре новых байта. Чтобы гарантировать, что каждый новый байт будет отличаться от другого (даже если все четыре байта те же самые), процесс сначала умножает каждый байт на различный набор констант и затем смешивает их. Смешивание может быть обеспечено матричным умножением. Как мы обсуждали в лекциях 2-3, когда мы умножаем квадратную матрицу на матрицу-столбец, результат — новая матрица-столбец. После того как матрица умножена на значения строки в матрице констант, каждый элемент в новой матрице зависит от всех четырех элементов старой матрицы. Рисунок 9.11 иллюстрирует эту идею.

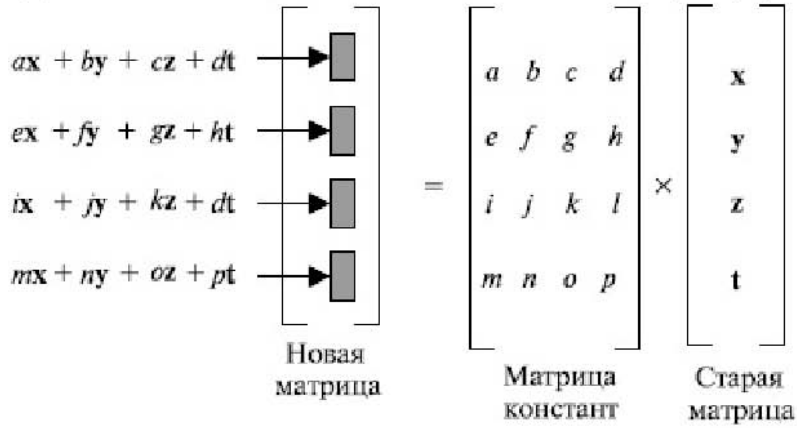


Рис. 9.11. Смешивание байтов с использованием смешивающей матрицы

AES определяет преобразование, называемое `MixColumns`. Для применения такого преобразования вводится также *обратное преобразование*, называемое `InvMixColumns`. Рисунок 9.12 показывает матрицу констант, используемую для этих преобразований. Эти две матрицы инверсны друг другу, когда элементы интерпретируются как слова из 8-ми битов (или полиномы) с коэффициентами в GF (2^8). Доказательство мы оставляем как упражнение.

Рис. 9.12. Матрица констант, используюемая MixColumns и InvMixColumns

MixColumns

Преобразование MixColumns работает на уровне столбца; оно преобразовывает каждый столбец матрицы состояний в новый столбец.

Это преобразование — фактически матричное умножение столбца матрицы состояний и квадратной матрицы констант. Байты в столбце матрицы состояний и в матрице констант интерпретируются как слова по 8 битов (или полиномы) с коэффициентами в GF (2). Умножение байтов выполняется в $GF(2^8)$ по модулю (10001101) или ($x^8 + x^4 +, x^3 + x + 1$). Сложение — это применение операции *ИСКЛЮЧАЮШЕЕ ИЛИ* (*XOR*) к словам по 8 бит. Рисунок 9.13 показывает преобразование MixColumns.

InvMixColumns

Преобразование InvMixColumns похоже на MixColumns-преобразование. Если две матрицы констант инверсны друг другу, то легко доказать, что эти два преобразования также инверсны друг другу.

Преобразования MixColumns и InvMixColumns инверсны друг другу.

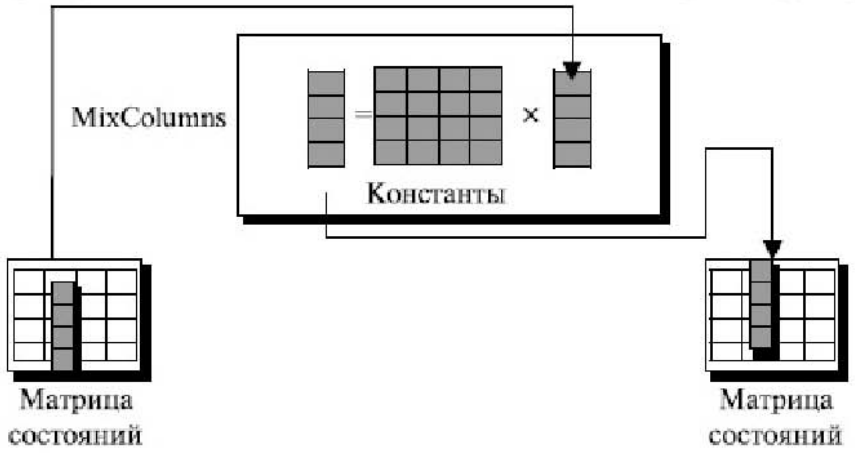


Рис. 9.13. Преобразование MixColumns

Алгоритм

Алгоритм 9.3. — программа для преобразования MixColumns.

```
MixColumns (S)
{
for (c == 0 to 3)
    mixcolumn (s_c)
}

mixcolumn (col)
{
    CopyColumn (col, t) //t is a temporary column

    col_0 ← (0 × 02) * t_0 ⊕ (0 × 03) * t_1 ⊕ t_2 ⊕ t_3
    col_1 ← t_0 ⊕ (0 × 02) * t_1 ⊕ (0 × 03) * t_2 ⊕ t_3
    col_2 ← t_0 ⊕ t_1(0 × 02) * t_2 ⊕ (0 × 03)t_3
    col_3 ← (0 × 03) * t_0 ⊕ t_1 ⊕ t_2 ⊕ (0 × 02) * t_3
}
```

Пример 9.3. Программа на псевдокоде для преобразования MixColumns

Алгоритмы `MixColumns` и `InvMixColumns` включают умножение и сложение в поле $GF(2^8)$. Как мы видели влекциях 5-6, есть простой и эффективный алгоритм для умножения и сложения в этом поле. Однако чтобы показать характер алгоритма (преобразование одного столбца одновременно), мы используем процедуру, называемую `mixcolumn`. Она может быть вызвана алгоритмом четыре раза. Процедура `mixcolumn` просто умножает строки матрицы констант на столбец в матрицы состояний. В вышеупомянутом алгоритме оператор `(•)`, используемый в процедуре `mixcolumn`, — умножение в поле $GF(2^8)$. Оно может быть заменено простой процедурой, как это уже рассматривалось в лекциях 5-6. Программу для `InvMixColumns` оставляем как упражнение.

Пример 9.5

Рисунок 9.14 показывает, как матрица состояний преобразуется, используя преобразование `MixColumns`. Рисунок также показывает,

что преобразование `InvMixColumns` создает первоначальный текст.

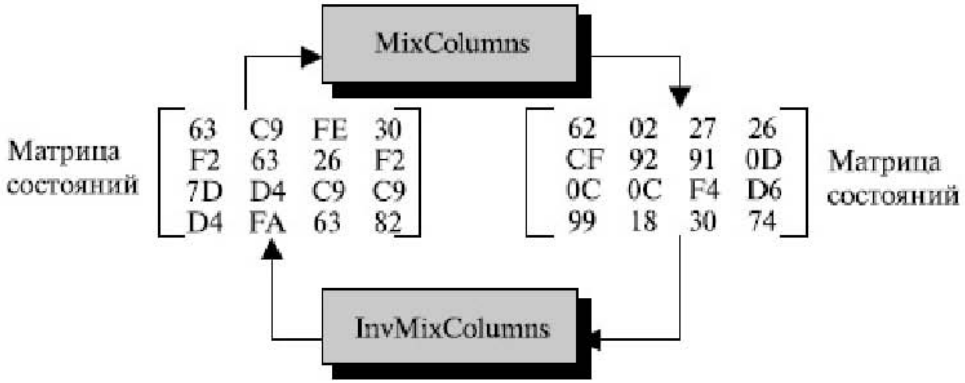


Рис. 9.14. Преобразование MixColumns в примере 9.5.

Обратите внимание, что байты, которые равны между собой в старой матрице состояний, больше не равны в новой матрице состояний. Например, два байта `F2` во второй строке изменены на `CF` и `0D`.

Добавление ключей

Вероятно, самое важное преобразование — это преобразование, которое включает ключ шифра, Все предыдущие преобразования используют известные алгоритмы, которые являются обратимыми. Если не добавлять ключевой шифр в каждом раунде, противник очень просто найдет исходный текст по данному ему зашифрованному тексту. В этом случае хранитель тайны Алисы и Боба — один единственный ключ шифра.

AES использует процесс, названный ключевым расширением (мы его обсудим позже), который из ключа шифра создает N_r+1 ключей раунда. Каждые ключи раунда — `128`-битовой длины и обрабатываются они как четыре слова по `32` бита. Для добавления ключа к матрице состояний каждое слово рассматривается как матрица-столбец.

AddRoundKey

AddRoundKey обрабатывает в один момент времени один столбец. Он подобен `MixColumns`. `MixColumns` умножает квадратную матрицу констант на каждый столбец матрицы состояний. *`AddRoundKey`* складывает ключевое слово раунда с каждым столбцом матрицы состояний. В `MixColumns` применяется матричное умножение; в *`AddRoundKey`* — операции сложения и вычитания. Так как сложение и вычитание в этом поле одни и те же, *`AddRoundKey`* инверсен сам себе. Рисунок 9.15 показывает преобразование *`AddRoundKey`*.

Преобразование AddRoundKey инверсно само себе.

Алгоритм

Преобразование *`AddRoundKey`* может быть представлено как операция *ИСКЛЮЧАЮШЕЕ ИЛИ* (*`XOR`*) каждой колонки матрицы состояний с соответствующим ключевым словом. Мы еще будем обсуждать, как расширяется ключ шифрования во множестве ключевых слов, но в данном случае мы определим преобразование, как это показано в алгоритме 9.4. Заметим, что s_c и $w_{round\ +4c}$ матрицы — колонки 4×1.

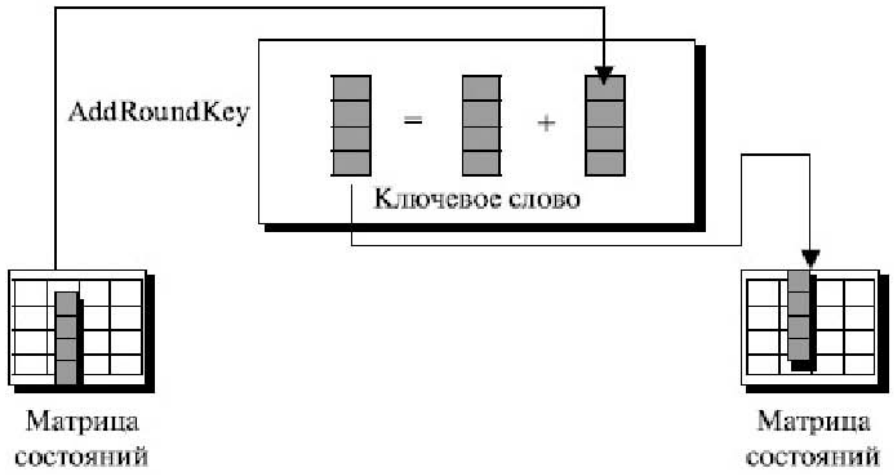


Рис. 9.15. Преобразование AddRoundKey

```
AddRoundKey (S)
{
for (c = 0 to 3)
```
$s_c \leftarrow s_c \oplus w_{round+4c}$
```
}
```

Пример 9.4. Преобразование AddRoundKey

Напомним, что оператор $\oplus$ здесь означает операцию *ИСКЛЮЧАЮШЕЕ ИЛИ* (*XOR*) для двух колонок матрицы, каждая из 4-х байт. Описание этой простой процедуры оставим для упражнений.

Усовершенствованный стандарт шифрования (AES — Advanced Encryption Standard)

В данной лекции особое внимание уделяется тому, как DES использует шифр Файстеля, чтобы достигнуть перемешивания и рассеивания на выходе из битов исходного текста к битам зашифрованного текста. Описывается процесс генерации ключей для раундов; и проводится анализ DES.

10.1. Расширение ключей

Для того чтобы создать ключ для каждого раунда, *AES* использует процесс ключевого расширения. Если номер раунда — N_r, процедура расширения ключей создает N_r+1 ключи раунда на 128 бит от единственного 128 -битового ключа шифра. Первые ключи раунда используются для преобразования перед раундом (*AddRoundKey*); остающиеся ключи раунда применяются для последнего преобразования (*AddRoundKey*) в конце каждого раунда.

Процедура расширения ключей создает слово за словом ключи раунда, где слово — массив из четырех байтов. Процедура создает $4 \times (N_r + 1)$ слова, которые обозначаются

$$w_0, w_1, w_2..., w_{4(Nr+1)-1}$$

Другими словами, в версии *AES*-128 (10 раундов) имеются 44 слова; в *AES*-192 версии (12 раундов), есть 52 слова; и в *AES* 256 версий (с 14 раундами) есть 60 слов. Каждый ключ раунда состоит из четырех слов. Таблица 10.1 показывает отношения между раундами и словами.

Таблица 10.1. Слова для каждого раунда

Раунд	Слова			
1	w_0	w_1	w_2	w_3
2	w_4	w_5	w_6	w_7
.....	.....			

N_r	w_{4Nr}	w_{4Nr+1}	w_{4Nr+2}	w_{4Nr+3}

Расширение ключей в AES-128

Посмотрим, как создаются ключи в версии *AES*-128; процессы для других двух версий за исключением небольших изменений — те же самые. Рисунок 10.1 показывает, как из исходного ключа получить 44 слова.

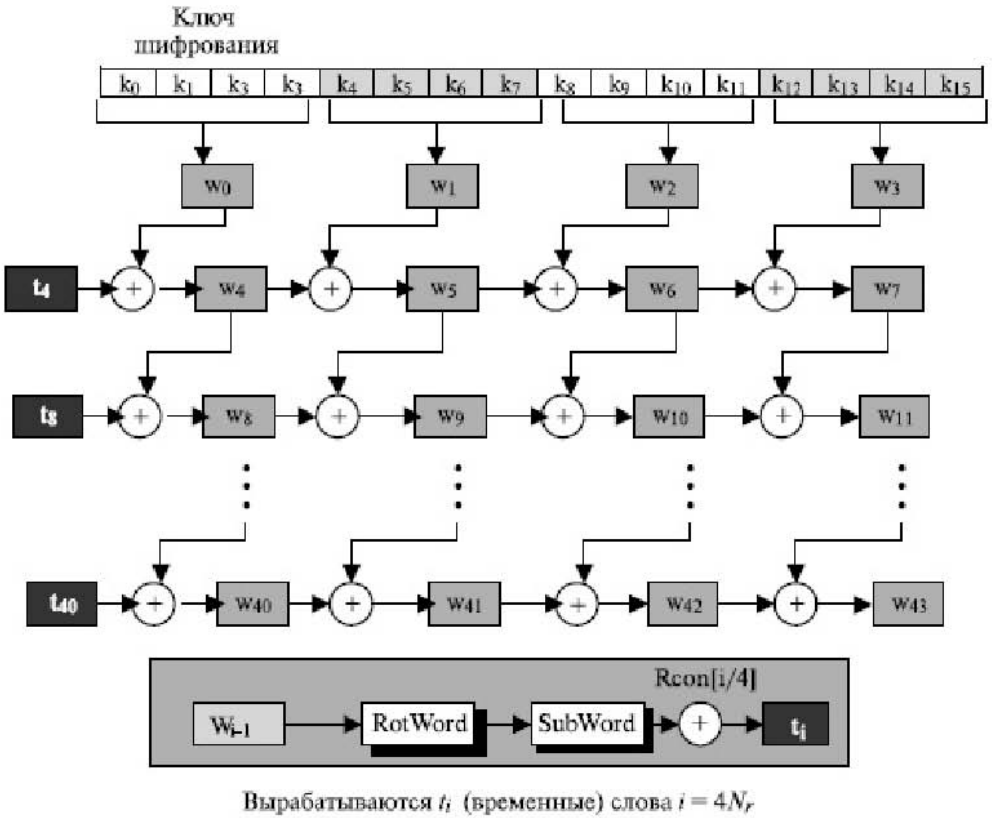


Рис. 10.1. Расширение ключей в AES

Процесс следующий:

1. Первые четыре слова (W_0, W_1, W_2, W_3) получены из ключа шифра. Ключ шифра представлен как массив из 16 байтов (k_0 до k_{15}).

Первые четыре байта (k_0 до k_3) становятся W_0 ; следующие четыре байта (k_4 до k_7) становятся w_1 ; и так далее. Другими словами, последовательное соединение (конкатенация) слов в этой группе копирует ключ шифра.

2. Остальная часть слов (w_i) от $i = 4 - 43$ получается следующим образом:

a. Если $(i \bmod 4) \neq 0$, $w_i = w_{i-1} \oplus w_{i-4}$, то согласно рисунку 10.1 это означает, что каждое слово получено из одного левого и одного верхнего.

b. Если $(i \bmod 4) = 0$, $w_i = t \oplus w_{i-4}$. Здесь t — временное слово, результат применения двух процессов, subword и rotword, со словом w_{i-1} и применения операции *ИСКЛЮЧАЮШЕЕ ИЛИ* с константой раунда R_{con}. Другими словами, мы имеем

$$t = SubWord(Rotword\ (w_{i-1})) \oplus RCon_{i/4}.$$

RotWord

RotWord (rotate word) — процедура, подобная преобразованию ShiftRows, но применяется только к одной строке. Процедура принимает слово как массив из четырех байт и сдвигает каждый байт влево с конвертированием.

SubWord

SubWord (*substitute word*) — процедура, подобная преобразованию *SubBytes*, но применяется только к одной строке. Процедура принимает каждый байт в слове и заменяет его другим.

RoundConstants

Каждая константа раунда R_{con} — это 4 -байтовое значение, в котором

самые правые три байта являются всегда нулевыми. Таблица 10.2 показывает значения для версии *AES*-128 (с 10 раундами).

Таблица 10.2. Константы RCon

Раунд	Константа (RCon)	Раунд	Константа (RCon)
1	$(01\ 00\ 00\ 00)_{16}$	6	$(20\ 00\ 00\ 00)_{16}$
2	$(02\ 00\ 00\ 00)_{16}$	7	$(40\ 00\ 00\ 00)_{16}$
3	$(04\ 00\ 00\ 00)_{16}$	8	$(80\ 00\ 00\ 00)_{16}$
4	$(08\ 00\ 00\ 00)_{16}$	9	$(1B\ 00\ 00\ 00)_{16}$
5	$(10\ 00\ 00\ 00)_{16}$	10	$(36\ 00\ 00\ 00)_{16}$

Процедура расширения ключей может использовать либо приведенную выше таблицу, когда вычисляет слова, либо поле GF(2^8), когда вычисляет крайние левые биты динамически, как это показано ниже.

$RC_1 \rightarrow x^{1-1} = x^0 \bmod prime = 1 \rightarrow 00000001 \rightarrow 01_{16}$

$RC_2 \rightarrow x^{2-1} = x^1 \bmod prime = x \rightarrow 00000010 \rightarrow 02_{16}$

$RC_3 \rightarrow x^{3-1} = x^2 \bmod prime = x^2 \rightarrow 00000100 \rightarrow 04_{16}$

$RC_4 \rightarrow x^{4-1} = x^3 \bmod prime = x^3 \rightarrow 00001000 \rightarrow 08_{16}$

$RC_5 \rightarrow x^{5-1} = x^4 \bmod prime = x^4 \rightarrow 00010000 \rightarrow 10_{16}$

$RC_6 \rightarrow x^{6-1} = x^5 \bmod prime = x^5 \rightarrow 0100000 \rightarrow 20_{16}$

$RC_7 \rightarrow x^{7-1} = x^6 \bmod prime = x^6 \rightarrow 01000000 \rightarrow 40_{16}$

$RC_8 \rightarrow x^{8-1} = x^7 \bmod prime = x^7 \rightarrow 10000000 \rightarrow 80_{16}$

$RC_9 \rightarrow x^{9-1} = x^8 \bmod prime = x^4+x^3+x+1 \rightarrow 00011011 \rightarrow 1B_{16}$

$RC_{10} \rightarrow x^{10-1} = x^9 \bmod prime = x^5+x^4+x^2+x \rightarrow 00110110 \rightarrow 36_{16}$

Крайний левый байт, который обозначен RC_i — это x^{i-1}, где i — номер раунда. *AES* использует неприводимый *полином* ($x^8 + x^4 + x^3 + x + 1$).

Алгоритм

Алгоритм 10.1 — простой алгоритм для процедуры расширения ключа (версия *AES*-128).

```
KeyExpansion ([key0 to key15], [w0 to w43]
{
for (i = 0 to 3)
wi ← key4i + key4i+1 + key4i+2 + key4i+3
 for (i = 4 to 43)
 {
   if (i mod 4 ≠ 0) wi ← wi−1 + wi−4
   else
   {
     t ← SubWord (RotWord (wi−1)) ⊕ RCon i/4
     wi ← t + wi−4          // t is a temporary word
   }
 }
}
```

Пример 10.1. Программа на псевдокоде для расширения ключей в AES128

Пример 10.1

Таблица 10.3 показывает, как вычисляются ключи для каждого раунда. Предполагается, что между Алисой и Бобом согласован ключ шифра на 128 битов — $(24\ 75\ A2\ B3\ 34\ 75\ 56\ 88\ 31\ E2\ 12\ 00\ 13\ AA\ 54\ 87)_{16}$.

Таблица 10.3. Пример расширения ключей

Раунд	Значения t	Первое слово в	Второе слово в	Третье слово в	Ч

Раунд	Значения t	раунде	раунде	раунде	сло
-		w_{00}=2475A2B3	w_{01}=34755688	w_{02}=31E21200	w_{03}=
1	AD20177D	w_{04}=8955B5CE	w_{05}=BD20E346	w_{06}=8CC2F146	w_{07}=
2	470678DB	w_{08}=CE53CD15	w_{09}=73732E53	w_{10}=FFB1DF15	w_{11}=
3	31DA48DO	w_{12}=FF8985C5	w_{13}=8CFAAB96	w_{14}=734B7483	w_{15}=
4	47AB5B7D	w_{16}=B822DEB8	w_{17}=34D8752E	w_{18}=479301AD	w_{19}=
5	6C762D20	w_{20}=D454F398	w_{21}=E08C86B6	w_{22}=A71F871B	w_{23}=
6	52C4F80D	w_{24}=86900B95	w_{25}=661C8D23	w_{26}=C1030A38	w_{27}=
7	E4133523	w_{28}=62833EB6	w_{29}=049FB395	w_{30}=C59CB9AD	w_{31}=
8	8CE29268	w_{32}=EE61ACDE	w_{33}=EAFE1F4B	w_{34}=2F62A6E6	w_{35}=
9	OA5E4F61	w_{36}=E43FE3BF	w_{37}=OEC1FCF4	w_{38}=21A35A12	w_{39}=
10	3PC6CD99	w_{40}=DBF92E26	w_{41}=D538D2D2	w_{42}=F49B88CO	w_{43}=

В каждом раунде вычисление последних трех слов очень просто. Для вычисления первого слова мы должны сначала вычислить значение временного слова `(t)`. Например, первое `t` (для раунда `1`) вычислено как

$$RotWord(13AA5487) = AA548713 \rightarrow SubWord(AA548713) = AC20177D$$
$$t = AC20177D \oplus Rcon_1 = AC20177D \oplus 01000000_{16} = AD20177D$$

Пример 10.2

Каждый ключ раунда в *AES* зависит от предыдущих ключей раунда. Зависимость, однако, нелинейная из-за преобразования `SubWord`. Сложение констант раунда также гарантирует, что каждый ключ раунда будет отличаться от предыдущего.

Пример 10.3

Два множества ключей раунда могут быть созданы от двух шифроключей, которые различаются только в одном бите.

Как показывает таблица 10.4, имеются существенные разности между двумя соответствующими ключами раунда — `R` означает "раунд", `B` и `D` означают разность битов.

Таблица 10.4. Сравнение двух множеств ключей

R.	Ключи для множества 1				Ключи д.	
-	1245A2A1	2331A4A3	B2CCAA34	C2BB7723	1245A2A1	2331A4A
1	F9B08484	DA812027	684D8A13	AAF6FD30	F9B08484	DA81202
2	B9E48028	6365AOOF	OB282A1C	A1DED72C	B9008028	6381AOC
3	AOEAF11A	C38F5115	C8A77B09	6979AC25	3DOEF11A	5E8F5115
4	1E7BCEE3	DDF49FF6	1553E4FF	7C2A48DA	839BCEA5	DD149FE
5	EB2999F3	36DD0605	238EE2FA	5FA4AA20	A2C910B5	7FDD8F0
6	82852E3C	B4582839	97D6CAC3	C87260E3	CB5AA788	B487288I
7	82553FD4	360D17ED	A1DBDD2E	69A9BDCD	588A2560	ECODOI
8	D12F822D	E72295CO	46F948EE	2F50F523	OB9F98E5	E7929508
9	99C9A438	7EEB31F8	38127916	17428C35	F2794CFO	15EBD9F
10	83AD32C8	FD460330	C5547A26	D216F613	E83BDABO	FDD0034

Пример 10.4

Понятие слабых ключей, которое мы обсуждали для *DES* в лекции 8, не применимо к *AES*. Предположим, что все биты в ключе шифра — нули. Ниже для этого случая показаны слова для некоторых раундов:

Перед раундом:	00000000	00000000	00000000	00000000
Round 01:	62636363	62636363	62636363	62636363
Round 02:	9B9898C9	F9FBFBAA	9B9898C9	F9FBFBAA
Round 03:	90973450	696CCFFA	F2F45733	OBOFAC99
.....	.....	.....	.....	.....
Round 10:	B4EF5BCB	3E92E211	23E951CF	6F8F188E

Все слова перед раундом и в первом раунде равны между собой. Во втором раунде первое слово соответствует третьему; второе слово соответствует четвертому. Однако после второго раунда совпадений нет; каждое слово различно.

нет; каждое слово различно.

Расширение ключа в AES-192 и AES-256

Алгоритмы расширения ключей в *AES*-192 и *AES*-256 — очень похожи на алгоритм расширения ключа в *AES*-128, со следующими отличиями:

1. В *AES*-192 слова сгенерированы в группы по шесть вместо четырех.

a. Ключ шифра создает первые шесть слов (w_0 к w_5).

b. Если $i \bmod 6 \neq 0$, то $w_i \leftarrow w_{i-1} + w_{i-6}$; иначе $w_i \leftarrow t + w_{i-6}$.

1. В *AES*-256 слова сгенерированы в группы по восемь вместо четырех.

a. Ключ шифра создает первые восемь слов (w_0 до w_7).

b. Если $i \mod 8 \neq 0$, то $w_i \leftarrow w_{i-1} + w_{i-8}$; иначе, w_i <-t + w_{i-8}.

c. Если i mod 4=0, но $i \mod 8 \neq 0$, то w_i = Subword (w_{i-1}) + w_{i-8}.

Анализ расширения ключа

Механизм расширения ключа в *AES* был разработан так, чтобы обеспечить несколько свойств, которые срывают действия криптоаналитика по раскрытию сообщения.

1. Даже если Ева знает только часть ключа шифра или значения слов в некотором ключевом раунде, этого недостаточно: она еще должна найти остальную часть ключа шифра, прежде чем сможет найти ключи всех раундов. Это обеспечивается нелинейностью процесса расширения ключа, полученной с помощью

отличаются по крайней мере в нескольких раундах.

3. Каждый бит ключа шифра изменяется в нескольких раундов. Например, изменение единственного бита в ключе шифра изменяет некоторые биты в нескольких раундах.
4. Использование констант, `RCons`, удаляет любую симметрию, которая может быть создана другими преобразованиями.
5. В отличие от *DES* в *AES* отсутствуют любые слабые ключи.
6. Процесс расширения ключа может быть легко реализован на всех платформах.
7. Процедура расширения ключа может быть реализована без применения отдельных таблиц; вычисления могут быть сделаны с использованием полей $GF(2^8)$ и $FG(2)$.

10.2. Шифры

Теперь посмотрим, как *AES* использует четыре типа преобразований для шифрования и *дешифрования*. В стандарте *алгоритм шифрования* упоминается как шифр и алгоритм *дешифрования* — как обратный шифр.

Как мы упоминали прежде, ADVANCED ENCRYPTION STANDARD — шифр не-Файстеля, а это означает, что каждое преобразование или группа преобразований должны быть обратимыми. Кроме того, шифр и обратный шифр должны использовать эти операции таким способом, чтобы они отменяли друг друга. Ключи раунда должны использоваться в обратном порядке. Ниже приведены два различных проекта, которые могут быть использованы для различных реализаций. Мы обсудим оба проекта для *AES*-128; для других версий применяются те же самые проекты.

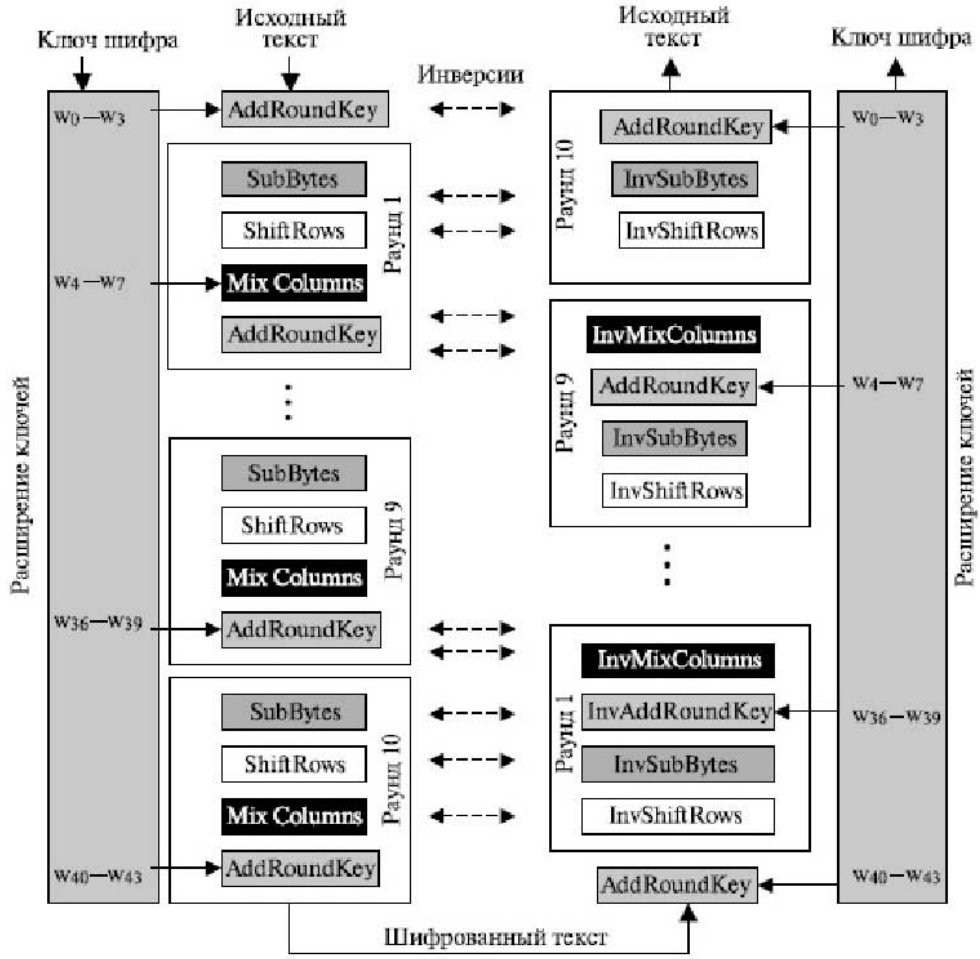


Рис. 10.2. Шифр и обратный шифр начального проекта

Первоначальный проект

В первоначальном проекте порядок преобразований в каждом раунде в шифре и обратном шифре не совпадает. Рисунок 10.2 показывает эту версию.

Во-первых, в обратном шифре изменяется порядок следования *SubBytes* (InvSubBytes) и ShiftRows (InvShiftRows). Во-вторых, в обратном шифре изменен порядок выполнения

`MixColumns` и *`AddRoundKey`*. Эти изменения в порядке необходимы, чтобы в обратном шифре сделать порядок работы *обратных преобразований* инверсным по отношению к прямому шифру. Следовательно, алгоритм *дешифрования* в целом — *инверсия* алгоритма шифрования. Мы показали только три раунда, но остальные имеют тот же самый вид. Обратите внимание, что ключи раунда используются в измененном порядке. Обратите также внимание, что алгоритмы шифрования и *дешифрования* в первоначальном проекте не совпадают.

Алгоритм

Код для версии *AES*-128 этого проекта показан в алгоритме 10.2. Код для обратного шифра оставляем как упражнение.

```
Cipher( InBlock[16], OutBlock[16], w[0...43])

{
BlockToState (InBlock, S)

S ← SubBytes (S)
for (round = 1 to 10)
{
S ← ShiftRows (S)
If (round ≠ 10) S ← MixColumns (S)
S ← AddRoundKey (S, w[4 x round, 4 x round + 3])
}
StateToBlock (S, OutBlock);
}
```

Пример 10.2. Алгоритм 10.2. Программа на псевдокоде для шифра первоначального проекта

Альтернативный проект

Для тех приложений, которые предпочитают совпадающие алгоритмы для шифрования и *дешифрования*, был разработан обратный шифр. В

такой версии преобразования в обратном шифре перестроены так, чтобы сделать порядок преобразований тем же самым в прямом шифре и обратном шифре. В этом проекте обеспечена обратимость для пары преобразований, а не для каждого одиночного преобразования.

Пары SubBytes/ShiftRows

`SubBytes` изменяет содержание каждого байта, не изменяя *порядок байтов* матрицы состояний; `ShiftRows` изменяет *порядок байтов* в матрице состояний, не изменяя содержание байтов. Это подразумевает, что мы можем изменить порядок этих двух преобразований в обратном шифре, не затрагивая обратимость целого алгоритма; рисунок 10.3 иллюстрирует идею. Обратите внимание, что комбинации двух преобразований — в шифре и обратном шифре — инверсны друг другу.

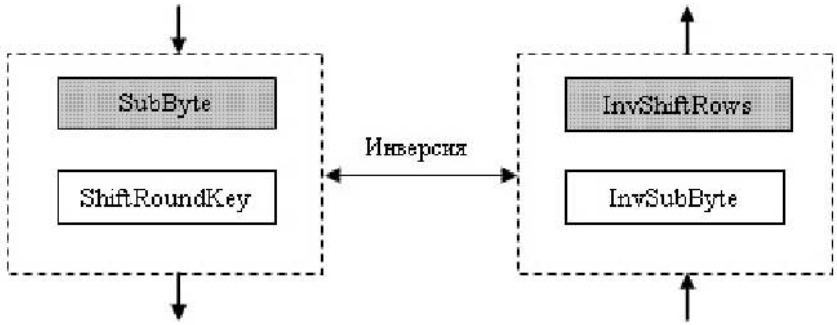


Рис. 10.3. Обратимость совокупности SubByte и SiftRows

Пара MixColumns/AddRoundKey

Здесь применяются два преобразования, которые имеют различные свойства. Однако эти пары могут стать *инверсиями* друг друга, если мы умножим матрицу ключей на *инверсию* матрицы констант, используемой в преобразовании `MixColumns`. Мы называем новое преобразование InvAddRoundKey. Рисунок 10.4 показывает новую конфигурацию.

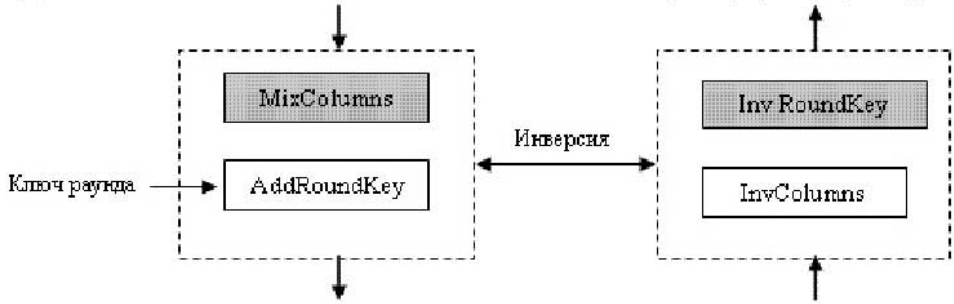


Рис. 10.4. Обратимость совокупности MixColumns и AddRoundKey

Можно доказать, что эти две комбинации теперь инверсны друг другу. В шифре мы обозначим входную матрицу состояний — `S` и выходную матрицу — `T`. В обратном шифре входная матрица — `T`. Ниже показано, что выходная матрица состояний — также `S`. Обратите внимание, что преобразование `MixColumns` — фактически произведение матрицы `C` (матрицы констант на матрицу состояний).

$$Cipher: T = CS \oplus K$$
$$InverseCipher: C^{-1} \oplus C^{-1\prime}K = C^{-1}(CS \oplus K) \oplus C^{-1}K = C^{-1}CS \oplus C^{-1}K \oplus C^{-1}K = S$$

Теперь мы можем показать шифр и обратный шифр для альтернативного проекта. Обратите внимание, что мы все еще должны использовать два преобразования *`AddRoundKey`* в *дешифровании*. Другими словами, мы имеем девять `InvAddRoundKey` и два *`AddRoundKey`* преобразования, как это показано на рис. 10.5.

Изменение алгоритма расширения ключей

Вместо того чтобы использовать преобразование `InvRoundKey` в обратном шифре, можно изменить алгоритм расширения ключей так, чтобы создавать различное множество ключей раунда для обратного шифра. Однако отметим, что ключ раунда для операций предварительного раунда и последнего раунда должен быть изменен. Ключи раунда от `1` до `9` необходимо умножить на матрицу констант. Этот алгоритм оставим для упражнений.

10.3. Примеры

В этом разделе приводятся некоторые примеры шифрования, *дешифрования* и *генерации ключей*, которые обсуждались в предыдущих разделах.

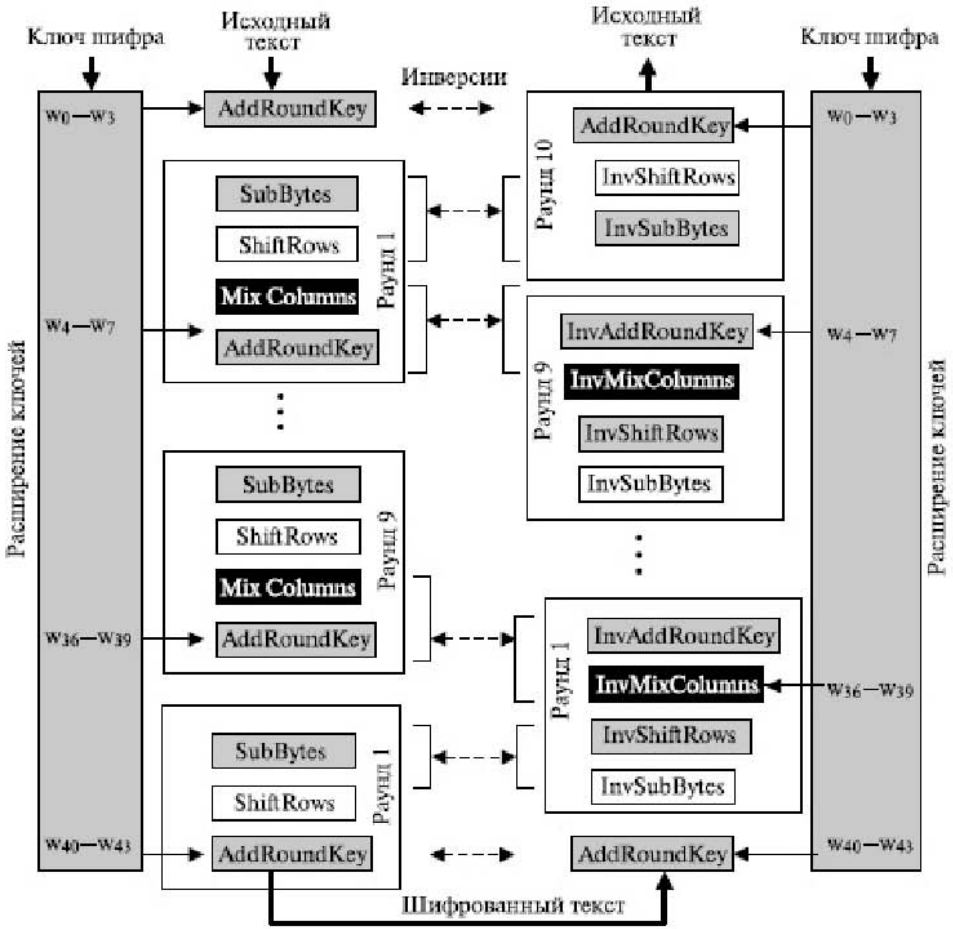


Рис. 10.5. Шифр и обратный шифр альтернативного проекта

Пример 10.5

Ниже показан блок зашифрованного текста, полученный из исходного текста с помощью *случайной выборки* ключей.

Исходный текст	00	04	12	14	12	04	12	00	0C	00	13	11	08	23	19	19
Ключ шифрования	24	75	A2	B3	34	75	56	88	31	E2	12	00	13	AA	54	87
Зашифрованный текст	BC	02	8B	D3	E0	E3	B1	95	55	0D	6D	FB	E6	F1	82	41

Таблица 10.5. показывает значения матрицы состояний и ключей раунда для этого примера.

Таблица 10.5. Пример шифрования

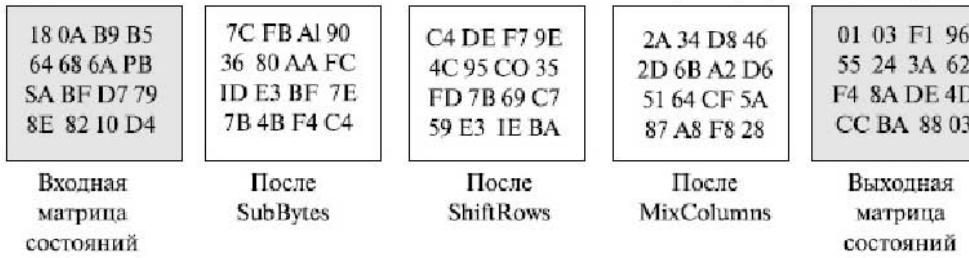
Раунд	Входная матрица состояний	Выходная матрица состояний	Ключ раунда
Предварительный раунд	00 12 *OC* 08	24 26 3D 1B	24 34 31 13
	04 04 00 23	71 71 E2 89	75 75 E2 AA
	12 12 13 19	BO 44 01 4D	A2 56 12 54
	14 00 11 19	A7 88 11 9E	B3 88 00 87
1	24 26 3D 1B	6C 44 13 BD	89 BD 8C 9F
	71 71 E2 89	Bl 9E 46 35	55 20 C2 68
	BO 44 01 4D	C5 B5 F3 02	B5 E3 F1 A5
	A7 88 11 9E	5D 87 *FC* 8C	CE 46 46 C1
2	6C 44 13 BD	1A 90 15 B2	CE 73 FF 60
	Bl 9E 46 35	66 09 ID *FC*	53 73 Bl D9
	C5 B5 F3 02	20 55 *5A* B2	CD 2E DF 7A
	5D 87 PC 8C	2B CB 8C 3C	15 53 15 D4
			FF 8C 73

			13
3	66 09 ID *FC*	1B 61 B4 B8	89 FA 4B 92
	20 55 *5A* B2	67 09 C9 45	85 AB 74 OE
	2B CB 8C 3C	4A 5C 51 09	C5 96 83 57
4	F6 7D A2 BO	CA E5 48 BB	B8 34 47 54
	1B 61 B4 B8	D8 42 AF 71	22 D8 93 01
	67 09 C9 45	Dl BA 98 2D	DE 75 01 OF
	4A 5C 51 09	4E 60 9E DF	B8 2E AD FA
5	CA E5 48 BB	90 35 13 60	D4 EO A7 F3
	D8 42 AF 71	2C FB 82 3A	54 8C IF IE
	Dl BA 98 2D	9E *FC* 61 ED	F3 86 87 88
	4E 60 9E DF	49 39 CB 47	98 B6 1B El
6	90 35 13 60	18 OA B9 B5	86 66 C1 32
	2C FB 82 3A	64 68 6A FB	90 1C 03 ID
	9E *FC* 61 ED	*5A* EF D7 79	OB 8D OA 82
	49 39 CB 47	8E B2 10 4D	95 23 38 D9
	18 OA B9 B5	01 63 F1 96	62 04 C5 F7

7	64 68 6A FB	55 24 3A 62	83 9F 9C 81
	5A EF D7 79	F4 8A DE 4D	3E B3 B9 3B
	8E B2 10 4D	CC BA 88 03	B6 95 AD 74
8	01 63 F1 96	2A 34 D8 46	EE *EA* 2F D8
	55 24 3A 62	2D 6B A2 D6	61 *FE* 62 E3
	F4 8A DE 4D	51 64 CF *5A*	AC IF A6 9D
	CC BA 88 03	87 A8 F8 28	DE 4B E6 92
9	2A 34 D8 46	OA D9 Fl 3C	E4 OE 21 F9
	2D 6B A2 D6	95 63 9F 35	3P Cl A3 40
	51 64 CF *5A*	2A 80 29 00	E3 *FC 5A* C7
	87 A8 F8 28	16 76 09 77	BF F4 12 80
10	OA D9 Fl 3C	BC EO 55 E6	*DB* D5 F4 OD
	95 63 9F 35	02 E3 OD Fl	F9 38 9B *DB*
	2A 80 29 00	8B Bl 6D 82	2E D2 88 4F
	16 76 09 77	D3 95 F8 41	26 D2 CO 40

Пример 10.6

Пример показывает матрицы состояний, раунд 7 в примере 10.5.



Пример 10.7

Один из курьезных случаев при рассмотрении шифрования — когда исходный текст состоит из одних нулей. Используем ключ шифра из примера 10.5 и получаем зашифрованный текст:

Исходный текст	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
Ключ шифрования	24	75	A2	B3	34	75	56	88	31	E2	12	00	13	AA	54	87
Зашифрованный текст	63	2C	D4	5E	5D	56	ED	B5	62	04	01	AO	AA	9C	2D	8D

Пример 10.8

Давайте проверим лавинный эффект, который мы обсуждали в лекции 8. Изменим только один бит в исходном тексте и сравним результаты. Мы изменили только один бит в последнем байте. Результат показывает эффект рассеивания и перемешивания. Изменение единственного бита в исходном тексте затронуло много бит в зашифрованном тексте.

Исходный текст 1	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
Исходный текст 2	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	01
Зашифрованный текст 1	63	2C	D4	5E	5D	56	ED	B5	62	04	01	AO	AA	9C	2D	8D
Зашифрованный текст 2	26	F3	9B	BC	A1	9C	0F	B7	C7	2E	7E	30	63	92	73	13

Пример 10.9

Ниже показан эффект использования ключа шифрования "все нули".

Исходный текст	00	04	12	14	12	04	12	00	0C	00	13	11	08	23	19	19
Ключ шифрования	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
Зашифрованный текст	5A	GF	4B	67	57	B7	A5	D2	C4	30	91	ED	64	9A	42	72

10.4. Анализ AES

Далее дан краткий обзор трех характеристик *AES*.

Безопасность

AES был разработан после *DES*. Большинство известных атак на *DES* было проверено на *AES*; ни одна из них до сих пор не нарушила безопасность *AES*.

Атака грубой силы

AES явно более безопасен, чем *DES*, из-за большего размера ключа (`128`, `192` и `256` битов). Давайте сравним *DES* с ключом шифра на `56` битов и *AES* с ключом шифра на `128` битов. Для *DES*, чтобы найти ключ, мы нуждаемся в 2^{56} испытаний (игнорируя проблему дополнения ключа); для *AES* — в 2^{128} испытаниях. Это означает, что если мы можем нарушить *DES* в `t` секунд, нам нужно ($2^{72} \times t$) секунд, чтобы нарушить *AES*. Это почти невозможно. Кроме того, *AES* обеспечивает две другие версии более *длинными ключами* шифра. Отсутствие слабых ключей — еще одно преимущество *AES* перед *DES*.

Статистические атаки

Сильное рассеивание и перемешивание, обеспеченное комбинацией преобразований *`SubByte`*, `ShiftRows` и `MixColumns`, удаляют любую частотную закономерность в исходном тексте. Многочисленные испытания не сумели выполнить *статистический анализ* зашифрованного текста.

Дифференциальные и линейные атаки

AES был разработан после *DES*. Дифференциальные и линейные атаки *криптоанализа* были, без сомнения, учтены. Подобные атаки на *AES* пока не обнаружены.

Реализация

AES может быть реализован в программном обеспечении, аппаратных средствах и программируемом оборудовании. Реализация может применять процесс поиска в таблице или процедуры, в которых использована четкая алгебраическая структура. Преобразование может быть ориентировано или на байт, или на слово. В ориентированной на байт версии алгоритм может использовать процессор на 8 битов; в ориентированной на слово версии — процессор на 32 бита. В любом случае, обработка делается очень быстро.

Простота и стоимость

Алгоритмы, используемые в *AES*, настолько просты, что могут быть легко реализованы с помощью дешевых процессоров и минимального объема памяти.

10.5. Рекомендованная литература

Нижеследующие книги и сайты дают более детальную информацию о предметах, рассмотренных в этой лекции. Пункты, приведенные в квадратных скобках, содержатся в списке в конце книги.

Книги

[Sta06] [Sti06] [Rhe03], [Sal03], [Mao04] и [TW06] рассматривают *AES*.

Сайты

Нижеследующие сайты дают больше информации о темах, рассмотренных в этой лекции.

- ссылка: csrc.nist.gov/publications/fips/ripsl97/fips-197.pdf - csrc.nist.gov/publications/fips/ripsl97/fips-197.pdf

- ссылка: http://www.quadibloc.com/crypto/co040401 .htm - http://www.quadibloc.com/crypto/co040401.htm
- ссылка: http: // www. ietef.org/rfc/rfc 3394. txt - http://www.ietef.org/rfc/rfc3394.txt

10.6. Итоги

- Усовершенствованный стандарт шифрования (*AES* — ADVANCED ENCRYPTION STANDARD) — стандарт *блочного шифра* с *симметричными ключами*, опубликованный Национальным Институтом стандартов *NIST* (National Institute of Standard and Technology) как Федеральный Стандарт Обработки Информации 197 (FIPST-197 — *FEDERAL* INFORMATION PROCESSING STANDARD 197). *AES* базируется на *Rijndael* алгоритме.
- *AES* — шифр не-Файстеля, который зашифровывает и расшифровывает блок данных длиной `128` битов. Оно использует `10`, `12` или `14` раундов.
- Размер ключа, который может быть `128`, `192` или `256` битов, зависит от числа раундов.
- *AES* ориентирован на работу с байтами. Исходный текст на `128` битов или зашифрованный текст рассматривается как шестнадцать байтов по `8` битов. Чтобы обеспечить выполнение некоторых математических преобразований на байтах, в *AES* определено понятие матрицы состояний. Матрица состояний — это матрица 4×4, в которой каждый вход является байтом.
- Чтобы обеспечить безопасность, *AES* использует четыре типа преобразований: подстановка, перестановка, смешивание и добавление ключа. Каждый раунд *AES*, кроме последнего, применяет эти четыре преобразования. Последний раунд использует только три из четырех преобразований.
- Подстановка определяется либо процессом поиска в таблице, либо математическим вычислением в поле $GF(2^8)$. *AES* использует два обратимых преобразования — *`SubBytes`* и `InvSubBytes`, которые являются *инверсиями* друг друга.
- Второе преобразование в раунде — сдвиг, которое переставляет байты. В шифровании преобразование названо `ShiftRows`, в

дешифровании — `InvShiftRows`. Преобразования `ShiftRows` и `InvShiftRows` инверсны друг другу.

- Преобразование смешивания изменяет содержание каждого байта, обрабатывая одновременно четыре байта и объединяя их, чтобы получить четыре новых байта. *AES* определяет два преобразования, `MixColumns` и `InvMixColumns`, используемые в шифровании и *дешифровании*. `MixColumns` умножает матрицу состояний на квадратную матрицу констант; `InvMixColumns` делает то же самое, используя обратную матрицу констант. Преобразования `MixColumns` и `InvMixColumns` инверсны друг другу.
- Преобразование, которое выполняет отбеливание, называется *AddRoundKey*. Предыдущая матрица состояний складывается (матричное сложение) с матричным ключом раунда, чтобы создать новую матрицу. Сложение отдельных элементов в этих двух матрицах выполняется в $GF(2^8)$, что означает, что над словами по 8 битов проводится операция *ИСКЛЮЧАЮШЕЕ ИЛИ* (*XOR*). Преобразование *AddRoundKey* инверсно само себе.
- В первой конфигурации (10 раундов с ключами на 128 битов) генератор ключей создает одиннадцать ключей раунда на 128 битов из ключа шифра на 128 битов. *AES* использует понятие слова для *генерации ключей*. Слова состоят из четырех байт. Ключи раунда генерируются слово за словом. *AES* нумерует слова от w_2 до w_{43}. Процесс называется "расширение ключа".
- Шифр *AES* для *дешифрования* использует два алгоритма. В первоначальном проекте порядок преобразований в каждом раунде — различен при шифровании и *дешифровании*. В альтернативном проекте преобразования в алгоритмах *дешифрования* перестроены так, чтобы порядок в шифровании и *дешифровании* был один и тот же. Во второй версии обратимость обеспечена для пары преобразований.

10.7. Набор для практики

Обзорные вопросы

1. Перечислите критерии, определенные *NIST* для *AES*.
2. Перечислите параметры (размер блока, размер ключа и *число раундов*) для трех версий *AES*.
3. Сколько преобразований имеется в каждой версии *AES*? Сколько ключей необходимо для каждой версии?
4. Сравните *DES* и *AES*. Какой из них ориентирован на работу с битом, а какой — на работу с байтом?
5. Определите матрицу состояний в *AES*. Сколько матриц состояний имеется в каждой версии *AES*?
6. Какие из четырех преобразований, определенных для *AES*, изменяют содержание байтов, а какие — не изменяют?
7. Сравните подстановку в *DES* и *AES*. Почему мы имеем только одну таблицу перестановки (`S` -блок) в *AES* и несколько — в *DES*?
8. Сравните перестановки в *DES* и *AES*. Почему надо иметь расширение и сжатие перестановки в *DES* и не надо — в *AES*?
9. Сравните ключи раунда в *DES* и *AES*. В каком шифре размер ключа раунда равен размеру блока?
10. Почему смешивающее преобразование (`MixColumns`) нужно в *DES*, но не нужно в *AES*?

Упражнения

1. При шифровании `S` -блоки могут быть или статическими, или динамическими. Параметры в статическом `S` -блоке не зависят от ключа.
    - Указать преимущества и недостатки статического и динамического `S` -блоков.
    - `S` -блоки в *AES* (таблицы подстановки), статические или динамические?
2. *AES* имеет больший размер блока, чем *DES* (`128` — в *AES* и `64` — в *DES*). Объясните, преимущество это или недостаток.
3. *AES* определяет различные реализации с различным числом раундов (`10, 12` и `14`); *DES* определяет только одну реализацию с `16` раундами. Объясните, преимущество это или недостаток *AES* и *DES* и в чем отличие.
4. *AES* определяет три различных размера ключа к шифру (`128,`

192 и 256); *DES* определяет только один размер ключа к шифру (56). Каковы преимущества и недостатки такого отличия?

5. В *AES* размер блока равен размеру ключей раунда (128 бит). В *DES* размер блока 64 бита, но размер ключей раунда — только 48 бит. Является ли эта разница преимуществом или недостатком *AES* по сравнению с *DES*?
6. Докажите, что преобразования ShiftRows и InvShiftRows — инверсны:
   - Покажите таблицу перестановки для ShiftRows. Таблица должна иметь 128 входов, но так как содержание байта не изменяется, таблица может иметь только 16 выходов; каждый выход представляет байт.
   - Повторите часть a для InvShiftRows преобразования.
   - Используя результаты частей a и b, докажите, что преобразования ShiftRows и InvShiftRows инверсны друг другу.
7. Используйте один и тот же ключ шифра, применив его для каждого из следующих преобразований на двух исходных текстах, которые отличаются только по первому биту. Найдите число изменившихся битов после каждого преобразования.
   - *SubBytes*
   - ShiftRows
   - MixColumns
   - *AddRoundKey* (с теми же самыми ключами раунда по вашему выбору)
8. Для того чтобы увидеть нелинейность преобразования *SubBytes*, покажите, что если a и b — два байта, то мы имеем
$$SubBytes(a \oplus b) \neq SubBytes(a) \oplus SubBytes(b)$$
Как пример используйте a = 0 x 57 и b = 0 x A2.
9. Дайте общую формулу для вычисления числа в каждом из видов преобразования *SubBytes*, ShiftRows, MixColumns и *AddRoundKey* и числа полных преобразований для каждой версии *AES*. Формула должна быть функцией числа раундов.
10. Измените рисунок 10.1 для *AES*-192 и *AES*-256.
11. Создайте две новые таблицы, которые показывают RCons константы для реализаций *AES*-192 и *AES*-256 (см. таблицу 10.2).

12. В *AES*-128 для предварительного раунда используются такие же ключи, как ключи шифрования. Справедливо ли это для *AES*-192? Справедливо ли это для *AES*-256?
13. На рисунке 9.8 перемножьте `X` и X^{-1} матрицы, чтобы доказать, что они инверсны друг другу.
14. Используя рисунок 9.12, перепишите квадратные матрицы `C` и C^{-1}, применяя полиномы с коэффициентами в `GF(2)`. Перемножьте эти две матрицы и докажите, что они являются обратными друг другу.
15. Докажите, что код в алгоритме 9.1 (преобразование *`SubByte`*) соответствует процессу, показанному на рис. 9.8.
16. Используя алгоритм 9.1 (преобразование *`SubByte`*), сделайте следующее:
    - Напишите код для процедуры, которая вычисляет *инверсию* байта в $GF(2^8)$.
    - Напишите код для `ByteToMatrix`.
    - Напишите код для `MatrixToByte`.
17. Напишите алгоритм для преобразования `InvSubBytes`.
18. Докажите, что код в алгоритме 9.2 (преобразование `ShiftRows`) соответствует процессу, показанному на рис. 9.9.
19. Используя алгоритм 9.2 (преобразование `ShiftRows`), напишите код для процедуры `copyrow`.
20. Напишите алгоритм для преобразования `InvShiftRows`.
21. Докажите, что код в алгоритме 9.3 (преобразование `MixColumns`) соответствует процессу, показанному на рис. 9.13.
22. Используя алгоритм 9.3 (преобразование `MixColumns`), напишите код для процедуры `copycolumns`.
23. Перепишите алгоритм 9.3 (преобразование `MixColumns`), заменив операторы `(.)` процедурой, называемой `multfield`, чтобы вычислить произведение двух байтов в поле $GF(2^8)$.
24. Напишите алгоритм для преобразования `InvMixColumn`.
25. Докажите, что код в алгоритме 9.4 (преобразование *`AddRoundKey`*), соответствует процессу, показанному на рис. 9.15.
26. В алгоритме 10.1 (расширение ключа):
    - Напишите кодовую программу для процедуры *`Subbyte`*.
    - Напишите код программу для процедуры `Rotword`.

27. Дайте два новых алгоритма для расширения ключа в *AES*-192 и *AES*-256 (см. алгоритм 10.1).
28. Напишите алгоритм расширения ключей для обратного шифра в альтернативном проекте.
29. Напишите алгоритм для обратного шифра в первоначальном проекте.
30. Напишите алгоритм для обратного шифра в альтернативном проекте.

Шифрование, использующее современные шифры с симметричным ключом

Эта лекция показывает, как концепции двух современных блочных шифров, рассмотренные в лециях 7-10, могут использоваться, чтобы зашифровать длинные сообщения. Она также вводит два новых понятия шифра потока.

11.1. Применение современных блочных шифров

Шифрование *симметричными ключами* может быть выполнено средствами современных *блочных шифров*. Два современных *блочных шифра*, обсужденные в лекциях 11 и 12,13, а именно DES и *AES*, разработаны для того, чтобы зашифровать и расшифровать блок текста фиксированного размера. DES зашифровывает и расшифровывает блок `64` битов; *AES* — блок `128` битов. В реальной жизни текст, который будет зашифрован, имеет переменный размер и обычно намного больший, чем `64` или `128` битов. Режимы работы были изобретены, чтобы зашифровать текст любого размера, используя либо DES, либо *AES*. Рисунок 11.1 показывает эти пять режимов работы, которые будет обсуждены далее.

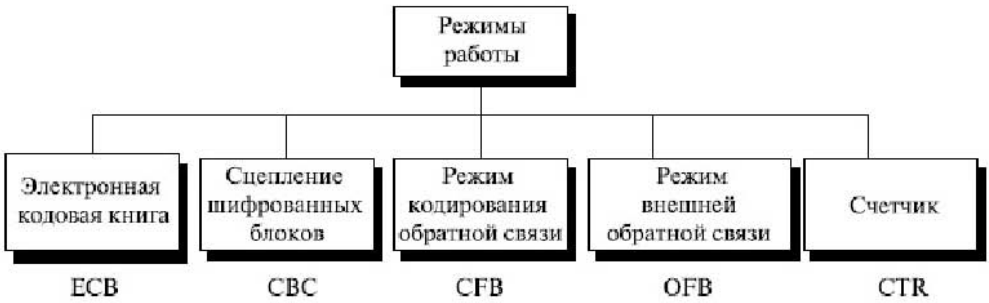


Рис. 11.1. Режимы работы

Режим электронной кодовой книги

Самый простой режим работы назван режимом электронной кодовой книги (ECB — ELECTRONIC CODEBOOK). Исходный текст разделен

на N блоков. Размер блока — n бит. Этот размер исходного текста не является кратным числом размера блока, текст дополняется, чтобы сделать последний блок по размеру таким же, как другие блоки. Один и тот же ключ используется, чтобы зашифровать и расшифровывать каждый блок. Рисунок 11.2 показывает шифрование и дешифрование в этом режиме.

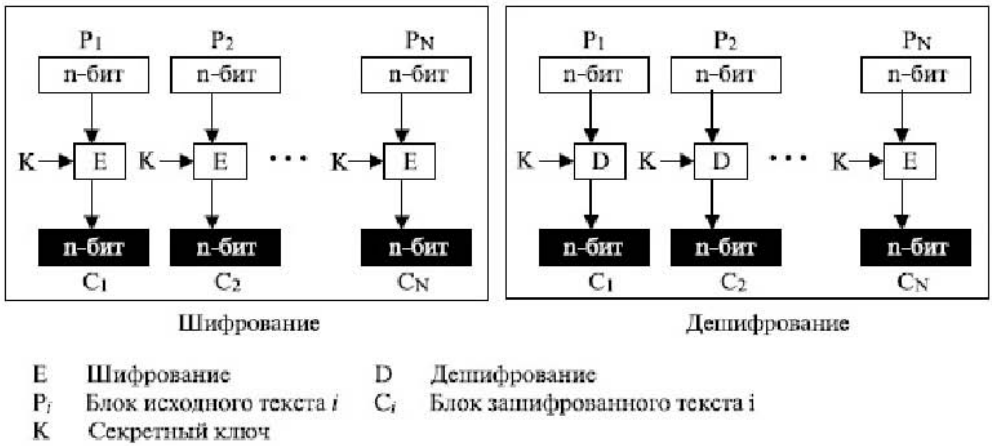


Рис. 11.2. Режим электронной кодовой книги (ECB)

Соотношение между исходным и зашифрованным текстами показано ниже:

Шифрование: $C_i = E_K(P_i)$

Дешифрование: $P_i = D_K(C_i)$

Пример 11.1

Этот пример показывает, как можно обеспечить, что каждый блок, посланный Алисой, может быть точно восстановлен на стороне Боба. Здесь используется то, что шифрование и дешифрование инверсны друг другу.

$$P_i = D_K (C_i) = P_i = D_K (E_K (P_i))$$

Пример 11.2

Этот режим называется режимом электронной кодовой книги, потому

что он может быть представлен 2^K кодовыми книгами (одной на каждый кодовый ключ). Каждая кодовая книга имеет 2^n входов и две колонки. Каждый вход сопоставляет исходному тексту соответствующий зашифрованный текст. Однако если K и n очень велики, кодовая книга будет слишком велика и ее компоновка и эксплуатация — неудобны.

Проблемы безопасности

В режиме *ECB* имеются следующие проблемы безопасности.

1. Образцы на уровне блока сохраняются. Например, одинаковые блоки в исходном тексте имеют одинаковый вид в соответствующих блоках зашифрованного текста. Если Ева узнает, что в зашифрованном тексте блоки 1, 5 и 10 одинаковы, она поймет, что блоки исходного текста 1, 5 и 10 — тоже одинаковые. Это —"дырочка" в безопасности. Например, Ева может выполнить исчерпывающий поиск и расшифровать только один из этих блоков, чтобы найти содержание всех их.
2. Независимость блоков создает Еве возможность для замены некоторых блоков зашифрованного текста без знания ключа. Например, если она знает, что блок 8 всегда передает некоторую заданную информацию, она может заменить этот блок соответствующим блоком в предварительно перехваченном сообщении.

Пример 11.3

Предположим, что Ева работает в компании месяц, несколько часов в неделю (ее ежемесячная оплата очень низкая). Она знает, что компания использует несколько блоков информации для каждого служащего, в котором седьмой блок является суммой денег, которая будет депонирована в учетной записи служащего. Ева может прервать зашифрованный текст, передаваемый банку в конце месяца, заменить блок информацией о ее оплате с копией блока с информацией об оплате полной рабочей недели её коллеги. Каждый месяц Ева может получать больше денег, чем она этого заслуживает.

Распространение ошибки

Единственный бит ошибки в передаче может создать ошибки в нескольких битах (обычно половине битов или всех битах) в соответствующих блоках. Однако ошибка не имеет никакого воздействия на другие блоки.

Алгоритм

Для шифрования или дешифрования могут быть написаны простые алгоритмы. Алгоритм 11.1 содержит процедуру, написанную в *псевдокоде* для шифрования. Процедуру для дешифрования оставляем как упражнение. EK зашифровывает только один единственный блок и может быть одним из шифров, рассмотренных в главах 6 или 7 (DES или *AES*).

```
ECB_Encryption (K,Plaintext bloks)
{
  for(i = 1 to N)
  {
    Ci <- EK (Pi),
  }
  return Cliphertext blocks
}
```

Пример 11.1. Режим шифрования ECB

Захват зашифрованного текста

В режиме *ECB* может потребоваться дополнение, которое добавляется к последнему блоку, если он содержит менее n бит. Такое дополнение не всегда возможно. Например, рассмотрим ситуацию, когда зашифрованный текст должен быть сохранен в буфере, где до этого был предварительно сохранен исходный текст. В этом случае исходный текст и зашифрованный текст должны быть одинаковой длины. Техника, которая называется захват зашифрованного текста (CTS — CipherText Stealing) позволяет использовать режим *ECB* без указанного выше

дополнения. В этой методике последние два блока исходного текста P_{N-1} и P_N, зашифрованы раздельно по-другому и в другом порядке, как показано ниже. Предположим, что P_{N-1} имеет n бит, а P_N имеет m бит, где $m < n$.

$$X = E_K(P_{N-1}) \rightarrow C_N = head_m(X)$$
$$Y = P_N | tail_{n-m}(X) \rightarrow C_{N-1} = E_K(Y)$$

где headm — функция, отделяющая крайние левые m бит, tailn-m — функция, отделяющая крайние правые n-m бит.

Приложения

Режим работы *ECB* не рекомендуется для шифрования сообщений, содержащих больше чем один блок, который будет передан через несекретный канал. Если сообщение достаточно коротко, чтобы передать его в одном блоке, проблемы безопасности и распространения ошибок терпимы.

Одна область, где независимость между блоками зашифрованного текста полезна, — это там, где информация будет зашифрована прежде, чем она будет сохранена в базе данных, или расшифрована прежде, чем она будет извлечена из памяти. Поскольку порядок шифрования и дешифрования блоков не важен, в этом режиме доступа к базе данных он может быть случаен, если каждая запись — блок или множество блоков. Запись может быть извлечена из середины, расшифрована, и зашифрована, не затрагивая другие записи. Другое преимущество этого режима — это то, что мы можем использовать *параллельную обработку*, если нужно, например, создать огромную зашифрованную базу данных.

Режим сцепления блоков шифрованного текста (CBC)

Следующая эволюция в работе режимов — режим сцепления блоков шифрованного текста (CBC — Cipher Block Chaining). В режиме *CBC* каждый блок исходного текста, прежде чем быть зашифрованным,

обрабатывается с помощью проведения операции ИСКЛЮЧАЮЩЕЕ ИЛИ с предыдущим блоком шифра. Когда блок зашифрован, блок передают, но копия сохраняется в памяти, которая используется в шифровании следующего блока. Читатель может задать вопрос о начальном блоке, поскольку перед первым блоком нет блока зашифрованного текста. В этом случае используется фальшивый блок, называемый вектор инициализации (IV). Передатчик и приемник согласуют заданный заранее `IV`. Другими словами, `IV` используется вместо несуществующего C_0. Рисунок 11.3 показывает режим *CBC*. На передающей стороне операция ИСКЛЮЧАЮЩЕЕ ИЛИ проводится перед шифрованием; а на стороне приемника дешифрование проводится перед операцией ИСКЛЮЧАЮЩЕЕ ИЛИ.

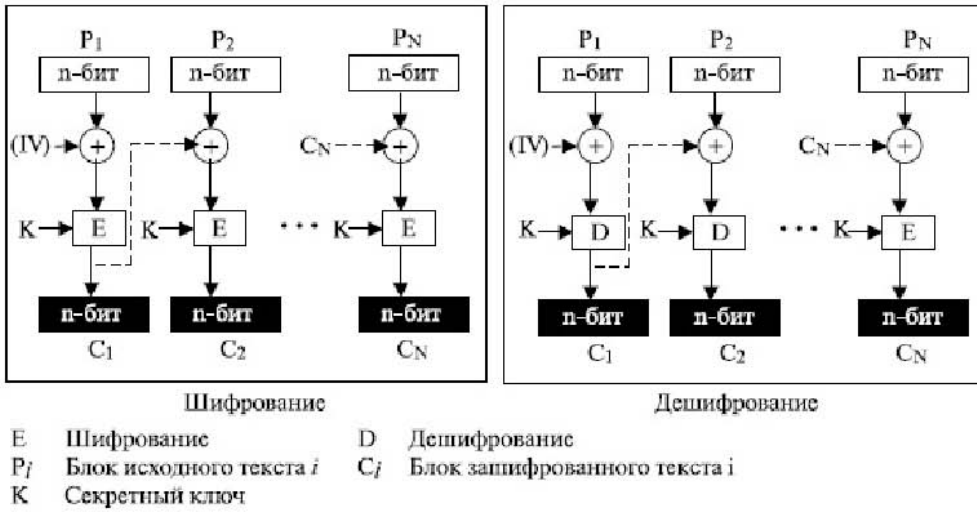


Рис. 11.3. Режим цепочки блоков шифротекста

Соотношение между исходным текстом и зашифрованным текстом показано ниже:

$$\text{Шифрование:}$$
$$C_0 = IV$$
$$C_i = E_K(P_i \oplus C_{i-1})$$
$$\text{Дешифрование:}$$
$$C_0 = IV$$
$$P_i = D_K(C_i) \oplus C_{i-1}$$

Пример 11.4

Можно доказать, что каждый блок исходного текста на стороне Алисы может быть точно восстановлен на стороне Боба — потому что шифрование и дешифрование инверсны друг другу.

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

Вектор инициализации (IV)

Вектор инициализации (IV) должен быть известен передатчику и приемнику. Хотя сохранение этого вектора в тайне не требуется, целостность вектора играет важную роль в безопасности режима *CBC* ; IV помогает сохранить безопасность изменения информации. Если Ева может изменить значения бит IV, это может изменить значения бит первого блока.

Для того чтобы использовать IV, рекомендовано несколько методов. Передатчик может выбрать *псевдослучайное число* и передать его через безопасный канал (например, использующий режим *ECB*). Фиксированное значение может быть согласовано Алисой и Бобом как IV, когда ключ засекречивания установлен. Это может быть часть ключа засекречивания, и так далее.

Проблемы безопасности

В режиме *CBC* имеются следующие две проблемы безопасности.

1. Одинаковые блоки исходного текста, принадлежащие тому же самому сообщению, зашифровываются в различные блоки зашифрованного текста. Другими словами, отдельные образцы в блоке не сохраняются. Однако когда два сообщения равны, их шифрованный текст одинаковый, если они используют те же самые IV. Фактически, если первые M блоков в двух различных сообщениях равны и IV совпадает, они будут зашифрованы в одинаковые блоки. По этой причине некоторые специалисты рекомендуют использование для IV метки времени.

2. Ева может добавить некоторые блоки зашифрованного текста в конце потока зашифрованного текста.

Распространение ошибки

В режиме *CBC* единственный бит ошибки в блоке C_j зашифрованного текста в процессе передачи — в процессе дешифрования может создать ошибку в большинстве битов блока P_j исходного текста. Однако эта одиночная ошибка изменяет только один бит в исходном тексте блока P_{j+1} (бит в том же самом местоположении). Доказательство этого факта оставляем как упражнение. Исходный текст блоков от P_{j+2} до P_N не затрагивается этим единственным битом ошибки. Единственный бит ошибки в зашифрованном тексте — самовосстанавливаемый.

Алгоритм

Алгоритм 11.2 дает *псевдокод* для шифрования — процедура encrypt. Она зашифровывает единственный блок (например, DES или *AES*). Алгоритм дешифрования оставляем как упражнение.

```
CBC_ Encryption (IV,K, Plaintext bloks)
{
  C0 <- IV
  for (i=1 to N)
  {
    Temp <- Pi ⊕ Ci-1
    Ci <- EK (Temp)
  }
return Ciphertext blocks
}
```

Пример 11.2. Алгоритм шифрования для режима ECB

Захват зашифрованного текста

Методика захвата зашифрованного текста, рассмотренная для режима *ECB*, может также быть применена к режиму *CBC*, как это показано ниже.

$$U = P_{N-1} \oplus C_{N-2} \rightarrow X = E_K(U) \rightarrow C_N = head_m(X)$$
$$V = P_N | pad_{n-m}(0) \rightarrow Y = X \oplus V \rightarrow C_{N-1} = E_K(Y)$$

Функция head — та же самая, что описана в режиме *ECB*; функция pad вставляет нули.

Приложения

Режим работы *CBC* может использоваться, чтобы зашифровать сообщения. Однако из-за механизма формирования цепочки *параллельная обработка* невозможна. Режим *CBC* не используется при шифровании и дешифровании массивов файлов с произвольным доступом, потому что шифрование и дешифрование требуют доступа к предыдущим массивам. Режим CBC применяется для установления подлинности сообщения.

Режим кодированной обратной связи (CFB)

ECB и режимы *CBC* предназначены для шифрования и дешифрования блоков сообщений. Размер блока, `n`, определяется принятым шифром. Например, `n = 64` для DES и `n = 128` для *AES*. В некоторых ситуациях мы должны использовать DES или *AES* как секретные шифры, но исходный текст или размеры блока зашифрованного текста должны быть меньшими. Например, чтобы зашифровать и расшифровывать символы `8` -битового ASCII, вы не захотели бы использовать один из традиционных шифров, обсужденных в лекции 4, потому что они ненадежны. Решение состоит в том, чтобы применить DES или *AES* в режиме кодированной обратной связи (CFB). В этом режиме размер блока, используемого в DES или *AES*, — `n`, но размер исходного текста или блока зашифрованного текста — `r`, где `r <n`.

Идея состоит в том, что DES или *AES* используются не для того, чтобы зашифровать исходный текст или расшифровывать зашифрованный

текст, а для того, чтобы зашифровать или расшифровывать содержание *регистра сдвига*, `S`, размером `n`. Шифрование сделано с применением операции ИСКЛЮЧАЮЩЕЕ ИЛИ к `r` -битовому блоку исходного текста с `r` -битовым *регистром сдвига*. Дешифрование сделано с применением операции ИСКЛЮЧАЮЩЕЕ ИЛИ к `r` -битовому блоку зашифрованного текста с `r` -битовым *регистром сдвига*. Для каждого блока *регистр сдвига* S_i выполняет сдвиг регистра S_{i-1} (предыдущий *регистр сдвига*) на `r` бит влево, заполняя самые правые `r` битов с C_{i-1}. Тогда S_i зашифрован в T_i. Только самые правые `r` битов T_i обрабатываются с помощью ИСКЛЮЧАЮЩЕГО ИЛИ с исходным текстом, из блока P_i получая C_i. Обратите внимание, что S_i, для первого блока — это `IV` — не сдвигается.

Рисунок 11.4 показывает режим *CFB* для шифрования; дешифрование то же самое, но роли блоков исходного текста (P_i) и блоков зашифрованного текста (C_i) меняются местами. Обратите внимание, что шифрование и дешифрование используют функцию шифрования основного *блочного шифра* (например, DES или *AES*).

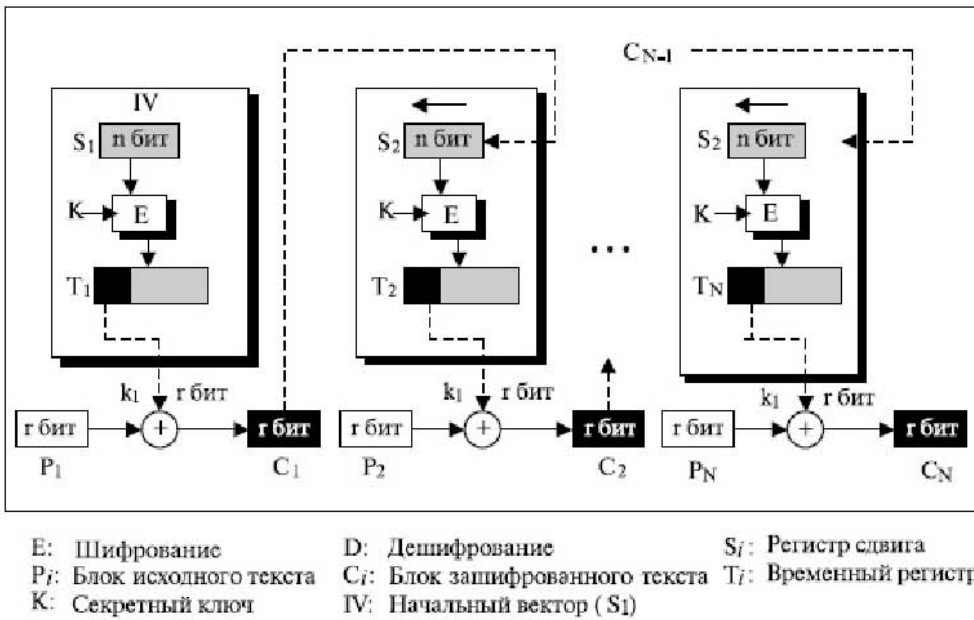


Рис. 11.4. Шифрование в режиме кодированной обратной связи

В режиме CFB шифрование и дешифрование используют функцию шифрования основного блочного шифра.

Соотношение между исходным текстом и блоками зашифрованного текста показано ниже:

$$\text{Шифрование}: C_i = P_i \oplus SelectLeft_r\{E_K[ShiftLeft_r(S_{i-1})|C_{i-1})]\}$$
$$\text{Дешифрование}: P_i = C_i \oplus SelectLeft_r\{E_K[ShiftLeft_r(S_{i-i})|C_{i-i})]\}$$

где `ShiftLeft` — процедура, которая сдвигает содержание ее параметра на `r` бит влево (крайние левые `r` -биты отбрасываются). Оператор | показывает конкатенацию (последовательное соединение). `SelectLeft` -процедура выбирает только крайние левые `r` -битов параметра. Возможно доказать, что каждый блок исходного текста на стороне Алисы может быть точно восстановлен на стороне Боба. Это доказательство оставляем как упражнение.

Интересно, что в этом режиме не требуется дополнение блоков, потому что размер блоков, `r`, обычно выбирается так, чтобы удовлетворить размеру блока данных, который нужно зашифровать (например, символ). Интересное также другое — что система не должна ждать получения большого блока данных (`64` бита или `128` битов) для того, чтобы начать шифрование. Процесс шифрования выполняется для маленького блока данных (таких как символ), Эти два преимущества приводят к двум недостаткам. *CFB* менее эффективен, чем *CBC* или *ECB*, потому что он применяет шифрование основным блочным шифром маленького блока размером `r`.

CFB как шифр потока

Хотя *CFB* — режим, предназначенный для того, чтобы использовать *блочные шифры*, такие как DES или *AES*, — он может быть шифром потока. Фактически, это несинхронный шифр потока, в котором ключевой поток зависит от зашифрованного текста. Рисунок 11.5 показывает процесс дешифрования и шифрования и генератор ключей.

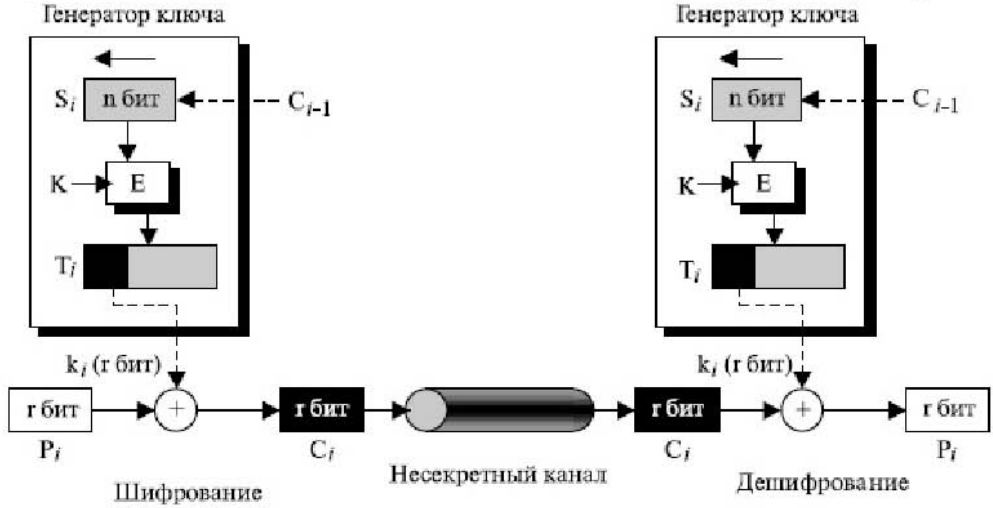


Рис. 11.5. Шифрование в режиме кодированной обратной связи как шифр потока

Рисунок 11.5 показывает, что основной шифр (DES или *AES*), ключ шифра (K) и предыдущий блок шифра (C_i) используются только для того, чтобы создать ключевые потоки (k_{i1} k_2..., k_N).

Алгоритм

Алгоритм 11.3 дает процедуру для шифрования. Он вызывает несколько других процедур, детали которых оставляем как упражнения. Обратите внимание, что мы написали алгоритм так, чтобы показать характер режима потока (обработка в реальном масштабе времени). Алгоритм выполняется, пока блоки исходного текста не будут зашифрованы.

```
CFB_Encryption (IV, K, r)
{
  i <- 1
  while (more blocks to encrypt)
  {
  input (P_i)
  (if i=l)
    S <- IV
  else
```

```
    {
    Temp <- shiftLeft(S)
    S <- concatenate (Temp, C_{i-1})
    }
  T <- E_K(S)
  k <- selectLeft_r(T)
  C_i <- P_i ⊕ k_i
  output (C_i)
  i <- i + 1
  }
}
```

Пример 11.3. Алгоритм шифрования кодированной обратной связи

Проблемы безопасности

В режиме *CFB* есть три первичных проблемы безопасности.

1. Точно так же как *CBC*, образцы на уровне блока не сохраняются. Одинаковые блоки исходного текста, принадлежащие одному и тому же самому сообщению, зашифровываются в различные блоки.
2. Одним и тем же ключом может быть зашифровано больше чем одно сообщение, но тогда значение `IV` должно быть изменено для каждого сообщения. Это означает, что Алиса должна использовать различные `IV` каждый раз, когда она передает сообщение.
3. Ева может добавить некоторый блок зашифрованного текста к концу потока зашифрованного текста.

Распространение ошибки

В *CFB* ошибка в единственном бите в зашифрованном тексте блока C_j в процессе передачи создает единственный бит ошибки (в той же самой позиции) в исходном тексте блока P_j. Однако большинство битов в следующих блоках исходного текста будут с ошибкой (с `50` -процентной вероятностью), пока биты C_j все еще находятся в *регистре сдвига*.

Вычисление числа затронутых блоков оставляем как упражнение. После того как *регистр сдвига* полностью регенерирован, система избавляется от ошибки.

Приложение

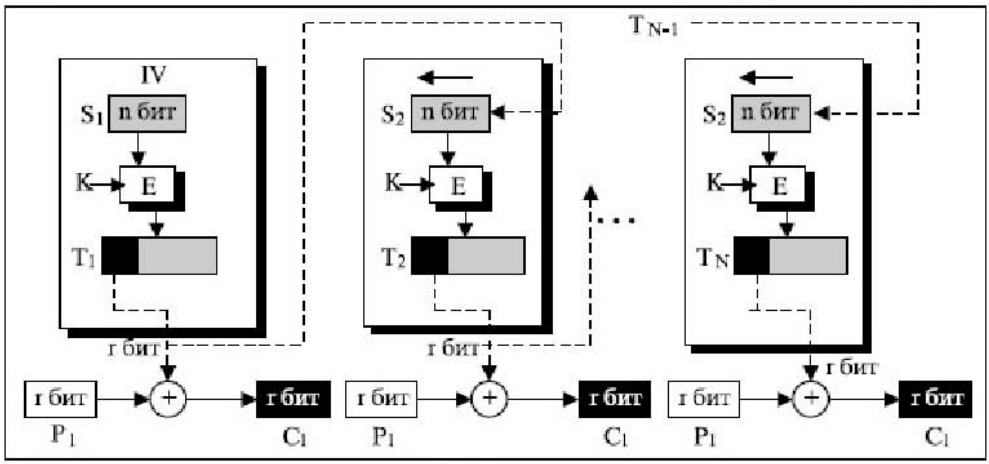
Режим работы *CFB* может использоваться, чтобы зашифровать блоки небольшого размера, такие как один символ или бит. Нет необходимости в дополнении, потому что размер блока исходного текста обычно устанавливается (`8` для символа или `1` для бита).

Специальный случай

Если блоки в тексте и в основном шифре — одного и того же размера (`n = r`), шифрование/дешифрование становится более простым, но построение диаграммы и алгоритм оставляем как упражнение.

Режим внешней обратной связи (OFB)

Режим внешней обратной связи (OFB — OUTPUT FEEDBACK) очень похож на режим *CFB*, с одной особенностью: каждый бит в зашифрованном тексте независим от предыдущего бита или битов. Это позволяет избежать распространения ошибок. Если при передаче возникает ошибка, она не затрагивает следующие биты. Подобно *CFB*, и передатчик и приемник используют алгоритм шифрования. Рисунок 11.6 показывает режим *OFB*.



Шифрование

E: Шифрование D: Дешифрование
P_i: Блок i исходного текста C_i: Блок i зашифрованного текста
K: Секретный ключ IV: Начальный вектор (S1)
S_i: Регистр сдвига T_i: Временный регистр

Рис. 11.6. Шифрование в режиме внешней обратной связи

OFB как шифр потока

OFB, так же как и *CFB*, может создать *поточный шифр* на базе основного шифра. Однако ключ потока не зависит от исходного текста или зашифрованного текста; значит, шифр потока — синхронный, как это обсуждалось в лекции 7. Рисунок 11.7 показывает шифрование и дешифрование, а также генератор ключей.

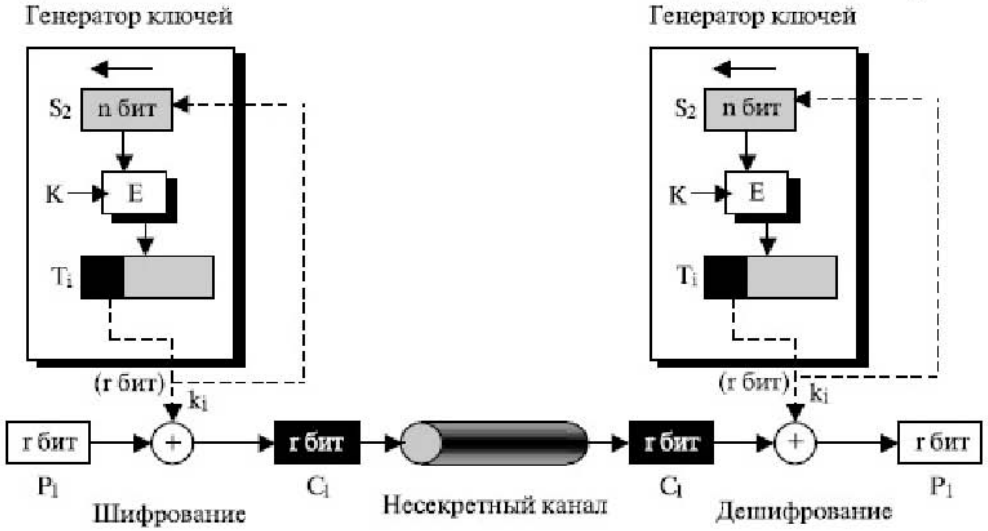


Рис. 11.7. Шифрование в режиме внешней обратной связи как шифрование потока

Алгоритм

Алгоритм 11.4 дает процедуру шифрования. Этот алгоритм последовательно вызывает другие процедуры, детали которых мы оставляем как упражнение. Заметим, что алгоритм написан так, чтобы показать режим потока (ситуация реального времени). Алгоритм работает, пока все блоки исходного текста не будут зашифрованы.

```
OFB_Encryption (IV, K, r)
{
 i <- 1
 while (more blocks to encrypt)
 {
 input (Pi)
 if (i=l)
  S <- IV
 else
  {
  Temp <- shiftLeftr(S)
  S <- concatenate (Temp, ki-1)
```

```
    }
  T <- Ek(s)
  k, <- selectLeftr (T)
  Ci <- Pi ⊕ ki
  output (Ci)
  i <- i + 1
  }
}
```

Пример 11.4.

Проблемы безопасности

В режиме *OFB* имеются следующие две проблемы безопасности.

1. Точно так же как в режиме *CFB*, образцы на уровне блока не сохраняются. Одинаковые блоки исходного текста, принадлежащие тому же самому сообщению, зашифровываются в различные блоки.
2. Любое изменение в зашифрованном тексте затрагивает исходный текст, зашифрованный в приемнике.

Распространение ошибки

Единственная ошибка в зашифрованном тексте затрагивает только соответствующий бит в исходном тексте.

Специальный случай

Если блоки в тексте и основном шифре имеют один тот же размер (`n = r`), шифрование/дешифрование становится более простым, но мы оставляем диаграммы и алгоритм как упражнение.

Режим счетчика (CTR)

В режиме счетчика (CTR — Counter) нет информации обратной связи. Псевдослучайный ключевой поток достигается с помощью счетчика. Счетчик на `n` бит инициализируется в заранее определенное значение (`IV`) и увеличивается по основному и заранее определенному правилу (`mod` 2^n). Чтобы обеспечивать случайность, величина приращения может зависеть от номера блока. Исходный текст и блок зашифрованного текста имеют один и тот же размер блока, как и основной шифр (например, DES или *AES*). Блоки размера `n` исходного текста зашифрованы так, чтобы создать зашифрованный текст с блоком размера `n`. Рисунок 11.8 показывает шифрование в режиме счетчика.

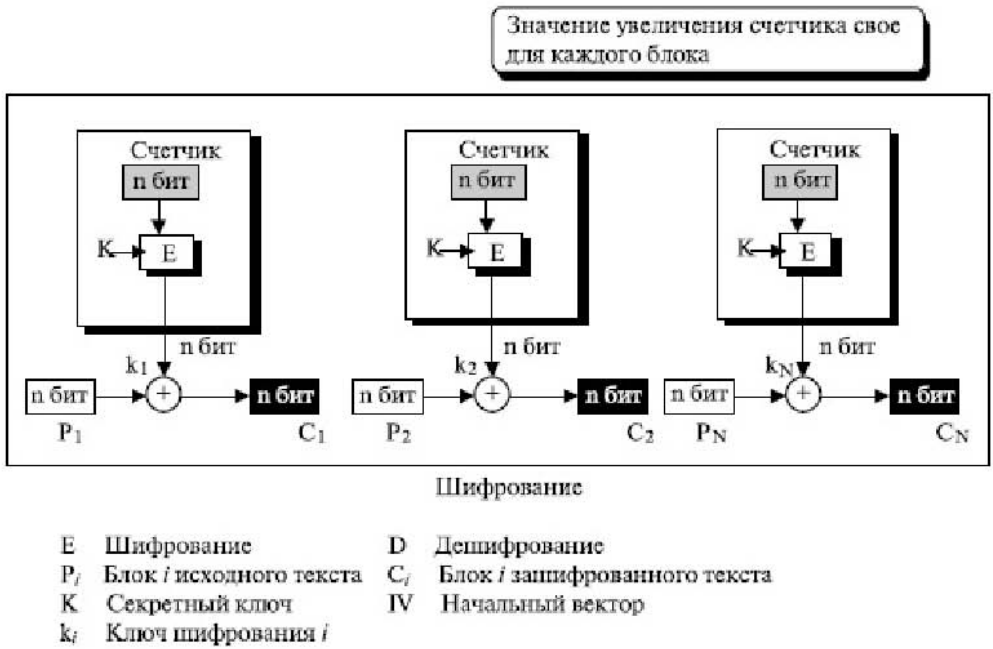


Рис. 11.8. Шифрование в режиме счетчика

Отношение между исходным текстом и блоками зашифрованного текста показано ниже.

$$\text{Шифрование} : C_i = P_i \oplus E_{ki}(\text{Счетчик})$$
$$\text{Дешифрование} : P_i = C_i \oplus E_{ki}(\text{Счетчик})$$

CTR использует функцию шифрования основного *блочного шифра* (E_K) и для шифрования, и для дешифрования. Достаточно легко доказать, что

блок P_i исходного текста может быть восстановлен из зашифрованного текста C_i. Это мы оставляем как упражнение.

Мы можем сравнить режим *CTR* с режимами *OFB* и *ECB*. Подобно *OFB*, *CTR* создает ключевой поток, который независим от предыдущего блока зашифрованного текста, но *CTR* не использует информацию обратной связи. Так же как *ECB*, *CTR* создает n -битовый зашифрованный текст, блоки которого независимы друг от друга — они зависят только от значений счетчика. Отрицательной стороной этого свойства является то, что режим *CTR*, подобно режиму *ECB*, не может использоваться для обработки в реальном масштабе времени. Алгоритм шифрования ждет перед шифрованием законченный n -разрядный блок данных. Положительная сторона этого свойства та, что режим, подобно режиму *ECB*, может использоваться, чтобы зашифровать и расшифровывать файлы произвольного доступа, и значение счетчика может быть связано номером записи в файле.

CTR как шифр потока

CFB, *OFB* и *CTR* — фактически шифры потока. Рисунок 11.9 показывает шифрование и дешифрование i-того блока данных.

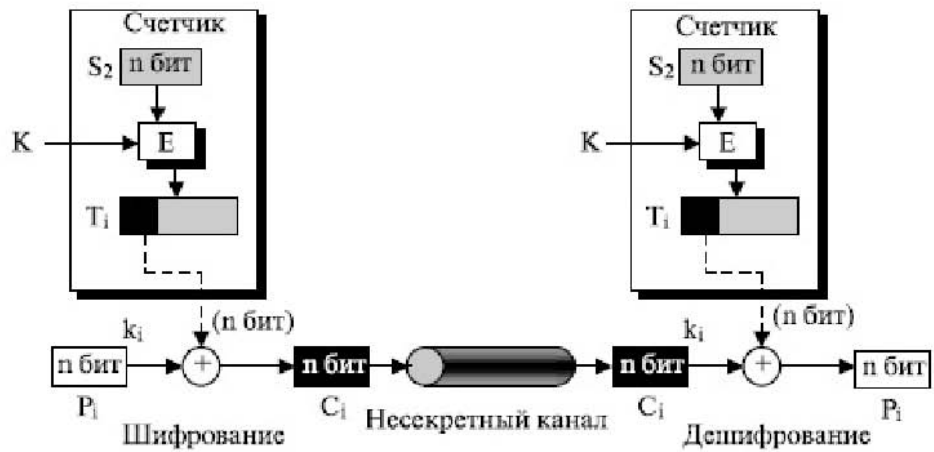


Рис. 11.9. Шифрование в режиме счетчика как шифр потока

Алгоритм

Алгоритм 11.5 содержит процедуру в *псевдокоде* для шифрования; алгоритм для дешифрования оставляем как упражнение. Здесь значение приращения зависит от номера блока. Другими словами, значения счетчика — `IV, IV + 1, IV + 3, IV + 6`, и так далее. Предполагается, что все `N` -блоки исходного текста готовы до начала шифрования, но алгоритм может быть переписан, чтобы избежать этого предположения.

Безопасность

Проблемы безопасности для режима *CTR* те же самые, что и для режима *OFB*.

Распространение ошибки

Единственная ошибка в зашифрованном тексте затрагивает только соответствующий бит в исходном тексте.

```
CTR_Encryption (IV, K, Plaintext blocks)
{
  Counter <-  IV
  for(i= 1 to N)
  {
    Counter <- (Counter + i - 1)mod 2^N
    k_i <- E_K (Counter)
    C_i <- P_i ⊕ k_i
   }
  return Ciphertext blocks
 }
```

Пример 11.5. Алгоритм шифрования CTR

Сравнение различных режимов

Таблица 11.1 сравнивает пять различных режимов работы, рассмотренных в этой лекции.

Таблица 11.1. Итоги режимов работы

Режим работы	Описание	Тип результата	Размер блока
ECB	Каждый n-битовый блок шифруется независимо тем же самым ключом	*Блочный шифр*	n
CBC	То же самое, что и в *ECB*, но каждый блок сначала складывается (ИСКЛЮЧАЮЩЕЕ ИЛИ) с предыдущим зашифрованным текстом	*Блочный шифр*	n
CFB	Каждый r-битовый блок складывается (ИСКЛЮЧАЮЩЕЕ ИЛИ) с r-битовым ключом, который является частью предыдущего текста шифра	Шифр потока	$r \leq n$
OFB	То же самое, что и в *CFB*, но *регистр сдвига* модифицирован с помощью предыдущего r-битового ключа	Шифр потока	$r \leq n$
CTR	То же самое, как в *OFB*, но счетчик используется вместо *регистра сдвига*	Шифр потока	n

11.2. Использование шифров потока

Хотя эти пять режимов работы допускают использование *блочных шифров* для шифрования сообщений или файлов в больших модулях (*ECB*, *CBC* и *CTR*) и маленьких модулях (*OFB* и *OFB*), иногда необходимо передать поток для того, чтобы зашифровать маленькие единицы информации — символы или биты. Шифры потока более эффективны для обработки в реальном масштабе времени. Некоторые шифры потока использовались в различных протоколах в течение прошлых нескольких десятилетий. Мы рассмотрим только два: *RC4* и A5/1.

RC4

RC4 — потоковый шифр, который был разработан в 1984 г. Рональдом Ривестом. *RC4* используется во многих системах передачи данных и

протоколах организации сети, например, SSL/*TLS* и *IEEE 802.11* (беспроводный стандарт LAN).

RC4 — байт-ориентированный шифр потока, в котором байт (8 битов) исходного текста складывается (ИСКЛЮЧАЮЩЕЕ ИЛИ) с байтом ключа, чтобы получить байт зашифрованного текста. Ключ засекречивания, из которого сгенерированы однобайтовые ключи в потоке ключей, может содержать от 1 до 256 байтов.

Матрица состояний

RC4 базируется на понятии матрицы состояний. В каждый момент матрица состояний 256 байтов активизируется, из нее случайным образом выбирается один байт, чтобы служить ключом для шифрования Идея может быть показана в виде массива байтов:

S [0] S [l] S [2] ••• S [255]

Заметим, что индексы диапазона элементов — между 0 и 255. Содержание каждого элемента — байт (8 битов), который может интерпретироваться как целое число от 0 до 255.

Идея

Рисунок 11.10 показывает идею *RC4*. Первые два блока выполняются только один раз (инициализация); перестановки для того, чтобы создавать ключ потока, повторяются, пока есть байты исходного текста, предназначенные для шифрования.

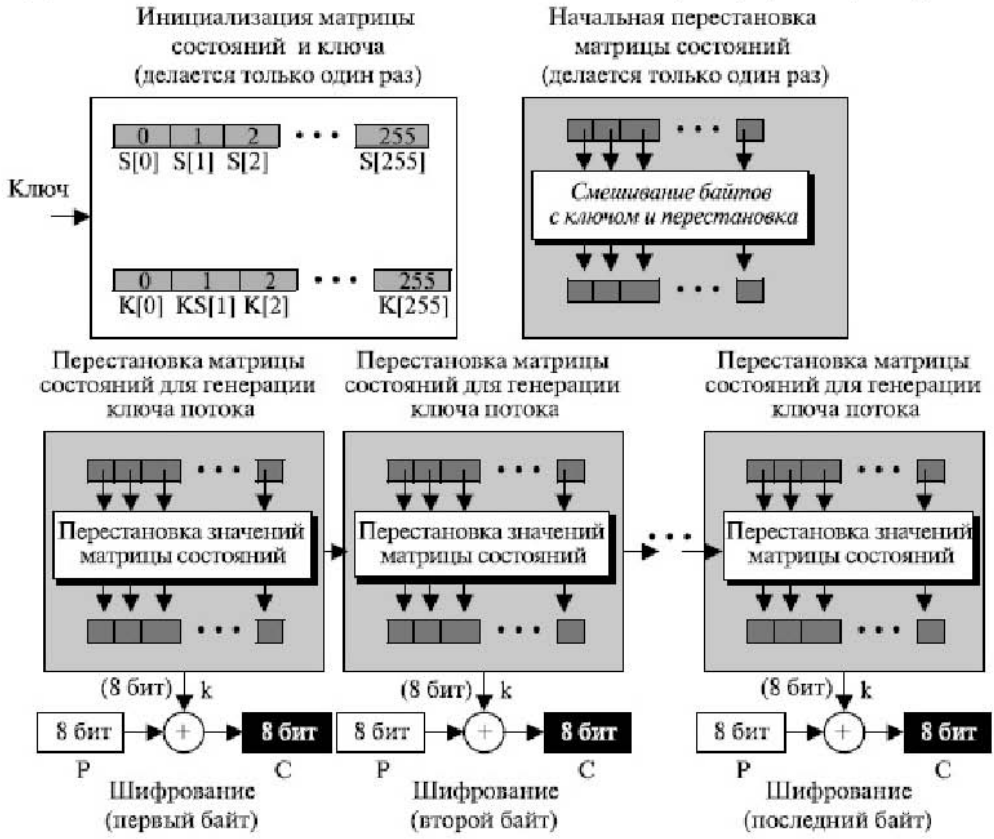


Рис. 11.10. Идея шифра потока RC4

Инициализация. Инициализация делается в два шага.

1. На первом шаге матрица состояний инициализируется для значений `0, 1..., 255`. Создается также массив ключей `K [0], K [1] ..., K [255]`. Если ключ засекречивания имеет точно `256` байтов, байты копируются в массив `K` ; иначе — байты повторяются, пока не заполнится массив `K`.

```
for (i = 0 to 255)
{
  S[i] <- i
  K[i] <- Key [i mod Key Length]
}
```

2. На втором шаге инициализированная матрица проходит перестановку (скрэмблирование элементов), основанную на значении байтов в `K[i]`. Ключевой байт используется только на этом шаге, чтобы определить, какие элементы должны быть заменены. После этого шага байты матрицы полностью перетасованы.

```
j <- 0
for (i = 0 to 255)
{
  j <- (1 + S[i] + K[i]) mod 256
  swap (S[i] , S[j])
}
```

Генерация ключевого потока. Ключи `k` в ключевом потоке генерируются один другим. Сначала элементы матрицы состояний переставляются на основе значений своих элементов и значений двух индивидуальных переменных `i` и `j`. Затем значения двух элементов матрицы состояний в позициях `i` и `j` используются, чтобы определить индекс элемента матрицы состояний, который служит как ключ `k`. Следующий код повторяется для каждого байта исходного текста, чтобы создать новый ключевой элемент в ключевом потоке. Переменные `i` и `j` инициализируются в `0` прежде, чем будет проведена первая итерация, но значение копируется от одной итерации к следующей.

```
i <- (i +1) mod 256
j <- (j +S[i]mod256
swap (S [i] , S[j])
k <- S [(S[i] + S[j]) mod 256]
```

Шифрование или дешифрование. После того как `k` были создан, байт исходного текста зашифровывается с помощью `k`, чтобы создать байт зашифрованного текста. Дешифрование представляет собой обратный процесс.

Алгоритм

Алгоритм 11.6 показывает процедуру, написанную на *псевдокоде*, для *RC4*.

```
RC4_Encryption (K)
{
  // Создание начальной матрицы состояний и ключевых байтов
  for (i = 0 to 255)
  {
    S[i] <- i
    K[i] <- Key [i mod Key Length]
  }
 // Перестановка байтов матрицы состояний на основе значений байта
  j <- 0
  for (i = 0 to 255)
  {
    j <- (j+ S[i] + K[i] mod 256
    замена (S[i] , S[j])
    //Непрерывная перестановка байтов, генерация ключей и шифровани
    i <- 0
    j <- 0
    while (пока есть байты для шифрования)
    i <- (i + 1) mod 256
    j <- (j +S[i]) mod 256
    swap(S[i],S[j])
    k <- S[(S[i]+S[j])mod256]
   //Ключ готов, шифрование
   input P
   C <- P ⊕ k
   output C
    }
}
```

Пример 11.6. Алгоритм шифрования для RC4

Пример 11.5

Чтобы показать случайность ключа потока, мы используем ключ засекречивания со всеми нулевыми байтами. Ключевой поток для 20 значений A: (222, 24, 137, 65, 163, 55, 93, 58, 138, 6, 30, 103, 87, 110, 146, 109, 199, 26, 127, 163).

Пример 11.6

Повторим пример 11.5, но пусть ключ засекречивания будет пять байтов `(15, 202, 33, 6, 8)`. Ключевой поток — `(248, 184, 102, 54, 212, 237, 186, 133, 51, 238, 108, 106, 103, 214, 39, 242, 30, 34, 144, 49)`. Снова случайность в ключевом потоке очевидна.

Проблемы безопасности

Известно, что шифр безопасен, если размер ключа — по крайней мере, `128` битов (`16` байтов). Это подтверждается сообщениями о некоторых атаках для малых размеров ключей (меньше, чем `5` байтов). Протоколы, которые сегодня использует *RC4*, устанавливают размеры ключей, которые делают *RC4* безопасным. Однако, как и для многих других шифров, рекомендуется, чтобы для различных сеансов использовались различные ключи. Это препятствует Еве использовать *дифференциальный криптоанализ* шифра.

A5/1

В этом разделе мы вводим шифр потока, который использует линейный *регистр сдвига* (см. лекции 9-10, *LFSR Linear Feed* Back *Shift Register*), чтобы создать битовый поток: A5/1. A5/1 (член семейства шифров A5) используется в Глобальной Системе Мобильной связи (*GSM*). Телефонная связь в *GSM* осуществляется как последовательность кадров на `228` битов, при этом каждый кадр длится `4,6` миллисекунды. A5/1 создает поток бит, исходя из ключа на `64` бита. Разрядные потоки собраны в буфере по `228` битов, чтобы складывать их по модулю два с кадром на `228` битов, как показано на рис. 11.11.

Рис. 11.11. Общий вид A5/1

Генератор ключей

A5/1 используются три *LFSR* на `19,22,23` бита. *LFSR* , содержащие биты символов и синхронизации показаны на рис 11.12.

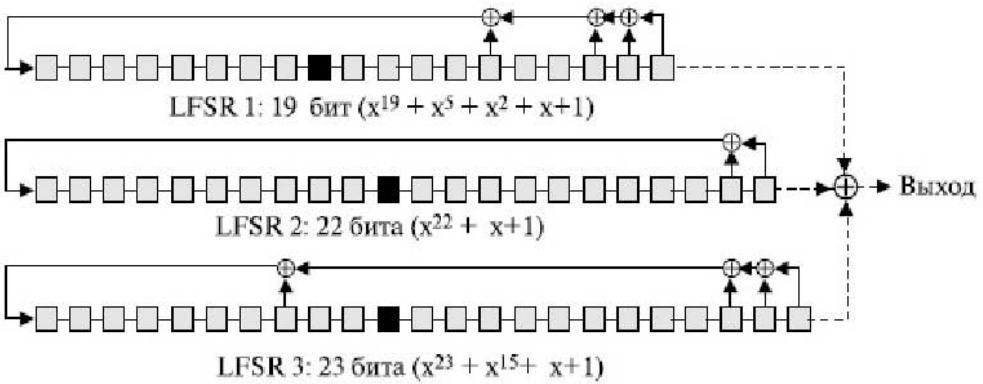


Рис. 11.12. Три линейных регистра сдвига для AS/5

Однобитовый выход обеспечивает тактовыми импульсами буфер на `228` битов, который используется для шифрования (или дешифрования).

Инициализация. Инициализация выполняется для каждого кадра шифрования (или дешифрования). Она использует ключ засекречивания на `64` бита и `22` бита соответствующего номера кадра. Следующие

шаги:

1. Сначала все биты в трех линейных регистрах сдвига устанавливаются в 0.

2. Второй: ключ на 64 бита смешивается со значением регистра согласно следующему коду. Каждый линейный регистр смещается на один шаг (синхронизация).

```
For (i = 0 to 63)
{
  Сложение по модулю 2 K[i] с крайними левыми битами всех трех реги
  Синхронизация всех трех линейных регистров сдвига
}
```

3. Повторить предыдущий процесс, но использовать 22 -битовый кадр.

```
for (i = 0 to 22)
{
  Сложение по модулю 2 номера кадра [i] с крайними левыми битами вс
  Синхронизация всех трех линейных регистров сдвига
}
```

4. В течение 100 циклов синхронизируется весь генератор. При этом используется мажоритарная функция (см. следующий абзац), для того чтобы определить, какой линейный *регистр сдвига* должен быть синхронизирован. Обратите внимание, что иногда синхронизация здесь означает, что два, а то и все три линейных *регистра сдвига* проходят процесс смещения.

```
for (i = 0 to 99)
{
  Синхронизация всего генератора, на основе мажоритарной функции
}
```

Мажоритарная функция. Значение мажоритарной функции (majority) с параметрами (b_1 b_2, b_3) равно 1, если значение большинства битов — 1 ; если это — 0, то ее значение — 0. Например, majority (1, 0, 1) = 1, но majority (0, 0, 1) = 0. Значение

мажоритарной функции определяется перед поступлением тактового импульса; три входных бита названы синхронизирующими битами: если самый правый бит равен нулю, это — биты линейных регистров `LFSR1 [10]`, `LFSR2 [11]` и `LFSR3 [11]`. Обратите внимание, что в литературе эти биты `8`, `10` и `10` отсчитывают слева (как это показано на рис. 11.12). Мы будем рассматривать `10`, `11` и `11`, считая справа. Это соглашение соответствует месту бита в характеристическом полиноме.

Ключевые биты потока. Генератор ключей создает ключевой поток в один бит при каждом тактовом импульсе. Прежде чем ключ создан, вычисляется мажоритарная функция. Затем каждый линейный *регистр сдвига* синхронизируется, если его бит синхронизации соответствует результату мажоритарной функции; иначе — он не синхронизируется.

Пример 11.7

В некоторый момент времени биты синхронизации — `1`, `0` и `1`. Какой должен быть *LFSR*?

Решение

Результат `Majority (1, 0, 1) = 1`. `LFSR1` и `LAFS3` сдвигаются, а `LFSR2` — нет.

Шифрование/дешифрование

Разрядные потоки, созданные генератором ключей, записываются в буфер, чтобы потом сформировать ключ на `228` битов, который затем складывает по модулю два с кадром исходного текста, чтобы создать кадр зашифрованного текста. В один момент времени делается шифрование/дешифрование одного кадра.

Проблемы безопасности

Хотя *GSM* продолжает использовать A5/1, уже были зарегистрированы несколько атак на *GSM*. Две из них были упомянуты. В 2000 году Алекс Бирюков, Дэвид Вагнер и Эди Шамир показали, что атака в реальном

масштабе времени находит ключ за несколько минут на основе известных малых исходных текстов, но это требует этапа предварительной обработки с 2^{48} шагами. В 2003 Экдахи и Джонсон (Ekdahi и Johannson) опубликовали атаку, которая вскрывала A5/1 за несколько минут, используя анализ исходного текста в течение 2–5 минут. Имея в виду некоторые новые атаки *GSM*, возможно, в будущем нужно будет сменить или укрепить A5/1.

11.3. Другие проблемы

Шифрование, которое использует блоки с *симметричными ключами* или шифры потока, требует обсуждения других проблем.

Управление ключами

Алиса и Боб должны совместно использовать секретный ключ, чтобы иметь надежную связь с использованием шифра с *симметричным ключом*. Если есть n объектов в сообществе, каждый из которых хочет связаться с n – 1 другим объектом, то тогда необходимы n (n – 1) ключей засекречивания. Однако при шифровании *симметричными ключами* один ключ может использоваться в обоих направлениях: от Алисы к Бобу и от Боба к Алисе. Это означает, что нужно только n (n – 1)/2 ключей. Если n — приблизительно миллион, то должны быть выданы почти пятьсот миллионов ключей. Поскольку это нереально, то были найдены несколько других решений. Первое: каждый раз, когда Алиса и Боб хотят связаться, они могут создать между собой сеансовый (временный) ключ. Второе: могут быть установлены один или более *центров распределения ключей*, чтобы распределять *сеансовые ключи* для объектов. Все эти проблемы — часть теории управления ключами.

Управление ключами будет обсуждаться в лекции 15.

Генерирование ключей

Другая проблема в шифровании *симметричными ключами* — *безопасная генерация* ключа. Различные шифры с *симметричным*

ключом нуждаются в ключах различных размеров. Выбор ключа должен базироваться на гарантии безопасности систематического метода для избежания утечки. Если Алиса и Боб генерируют *сеансовые ключи* между собой, они должны выбрать ключ случайным образом, так, чтобы Ева не могла предвидеть, каков будет следующий ключ. Если ключи должен распределять ключевой центр, они должны иметь случайный характер, чтобы Ева не могла получить ключ, назначенный Алисе и Бобу, из ключа, назначенного Джону и Еве. Это подразумевает, что нужен генератор случайных (или псевдослучайных) чисел. Поскольку обсуждение генератора случайных чисел включает некоторые темы, которые еще не были рассмотрены, изучение генераторов случайных чисел представлено в приложении К.

Генераторы случайных чисел будут обсуждаться в приложении К.

11.4. Рекомендованная литература

Нижеследующие книги и сайты дают более детальную информацию о предметах, рассмотренных в этой лекции. Пункты, приведенные в квадратных скобках, содержатся в списке в конце книги.

Книги

[Sch99], [Sta06], [PHS03], [Sti06], [MOV97] и [KPS02] рассматривают режимы работы. [Vau06] и [Sta06] дают полные сведения о шифрах потока.

Сайты

Нижеследующие сайты содержат больше информации о темах, обсужденных в этой лекции.

- ссылка: http: // en.wikipedia.org/wiki/Block_cipher_modes_of_operation - http://en.wikipedia.org/wiki/Block_cipher_modes_of_operation
- ссылка: http://www.itl.nist.gov/fipspubs/fip81.htm - http://www.itl.nist.gov/fipspubs/fip81.htm

- ссылка: en.wikipedia.org/wiki/A5/1 - en.wikipedia.org/wiki/A5/1
- ссылка: en.wikipedia.org/wiki/RC4 - en.wikipedia.org/wiki/RC4

11.5. Итоги

- В реальных приложениях зашифрованный текст имеет переменные размеры и обычно намного большие, чем размер блока, определенный для современных *блочных шифров*. Режимы работы были изобретены, чтобы зашифровать текст любого размера, который обслуживается современными *блочными шифрами*. В этой лекции были рассмотрены пять режимов работы.
- Самый простой режим работы называется режимом электронной кодовой книги (*ECB — ELECTRONIC* CODEBOOK). Исходный текст разделен на `N` блоков. Размер блока — `n` бит. Каждый блок использует для шифрования и дешифрования один и тот же ключ.
- В режиме сцепления блоков шифротекста (*CBC — Cipher* Block *Chaining*) каждый блок исходного текста прежде чем зашифровывать, складывают по модулю два с предыдущим блоком зашифрованного текста. Когда блок зашифрован, его передают, но его копия сохраняется в памяти, чтобы ее можно было использовать для шифрования следующего блока. Передатчик и приемник согласуют заранее заданный вектор инициализации (`IV`), чтобы складывать его по модулю два с первым блоком зашифрованного текста.
- Чтобы шифровать маленькие модули данных в реальном масштабе времени, применяется режим кодированной обратной связи (*CFB — CIPHER FEEDBACK*), *CFB* применяет стандартные *блочные шифры*, такие как DES или *AES*, *регистр сдвига*, но использует операцию сложения по модулю два, чтобы зашифровать или расшифровывать модули данных. Режим *CFB* использует *блочные шифры*, но в результате — это шифр потока, потому что каждый модуль данных зашифровывается своим ключом.
- Режим внешней обратной связи (*OFB*) очень похож на режим *CFB*, с одной разницей — каждый бит в зашифрованном тексте независим от предыдущего бита или битов. Это позволяет избежать распространения ошибки. Вместо того чтобы использовать предыдущий блок зашифрованного текста, *OFB*

берет предыдущий ключ как информацию обратной связи.

- В режиме счетчика (*CTR*) нет информации обратной связи. Псевдослучайность в потоке достигается с помощью счетчика. Счетчик на `n` битов инициализируется установкой заранее заданного значения (`IV`) и увеличивается по заранее заданному правилу.
- Чтобы зашифровать маленькие единицы данных, такие как символы или биты, были разработаны для испытаний несколько шифров потока. Эти шифры потока более эффективны для обработки в реальном масштабе времени. В этой лекции рассматривались только два шифра потока — *RC4* и A5/l.
- *RC4* — шифр потока, ориентированный на байт, в котором байт (`8` битов) исходного текста надо сложить по модулю два с байтом ключа, чтобы создать байт зашифрованного текста. Секретный ключ, из которого генерируются однобайтовые ключи в ключевом потоке, может содержать от `1` до `256` байтов. Ключевой генератор потока базируется на перестановке `256` байтов.
- A5/1 — шифр потока, используемый для мобильной телефонной связи. A5/1 создает поток бит из ключа на `64` бита, используя три линейных *регистра сдвига*.

11.7. Набор для практики

Обзорные вопросы

1. Объясните, почему необходимы режимы работы, если для шифровки используются современные *блочные шифры*.
2. Перечислите пять режимов работы, рассмотренных в этой лекции.
3. Определите *ECB* (*ELECTRONIC* CODEBOOK) и перечислите его преимущества и недостатки.
4. Определите *CBC* (*CIPHER* BLOCK *CHAINING*) и перечислите ее преимущества и недостатки.
5. Определите *CFB* (*CIPHER FEEDBACK*) и перечислите его преимущества и недостатки
6. Определите *OFB* (OUTPUT *FEEDBACK*) и перечислите его преимущества и недостатки.

7. Определите *CTR* и перечислите его преимущества и недостатки.
8. Разделите пять режимов работы на две группы: те, которые используют функции шифрования и дешифрования, — основные шифры (например, DES или *AES*), и те, которые используют только функцию шифрования.
9. Разделите пять режимов работы на две группы: те, которые требуют дополнение текста, и те, которые не требуют этого.
10. Разделите пять режимов работы на две группы: те, которые используют один и тот же ключ для шифрования всех блоков, и те, которые используют ключевой поток для шифровки блоков.
11. Объясните основные различия между *RC4* и A5/1. Какой из них использует линейный *регистр сдвига*?
12. Каков размер модуля данных в *RC4*? Каков размер модуля данных в A5/1?
13. Перечислите режимы работы, которые могут быть ускорены *параллельной обработкой*.
14. Перечислите режимы работы, которые могут использоваться для шифровки файлов произвольного доступа.

Упражнения

1. Покажите, почему режим *CFB* создает несинхронный шифр потока, а режим *OFB* создает синхронный.
2. Сколько блоков затрагивает единственный бит ошибки в передаче в режиме *CFB*?
3. В режиме *ECB* бит `17` в зашифрованном тексте блока `8` разрушен в течение передачи. Найдите возможные разрушенные биты в исходном тексте.
4. В режиме *CBC* биты `17` и `18` в зашифрованном тексте блока `9` в процессе передачи были разрушены. Найдите возможные разрушенные биты в исходном тексте.
5. В режиме *CFB* биты `3-6` в зашифрованном тексте блока `11` разрушены (`r = 8`). Найдите возможные разрушенные биты в исходном тексте.
6. В режиме *CTR* блоки `3` и `4` полностью разрушены. Найдите возможные разрушенные биты в исходном тексте.
7. В режиме *OFB* полный зашифрованный текст блока `11` разрушен

(`r = 8`). Найдите возможные разрушенные биты в исходном тексте.

8. Докажите, что исходный текст, используемый Алисой, может быть восстановлен Бобом в режиме *CFB*.
9. Докажите, что исходный текст, используемый Алисой, может быть восстановлен Бобом в режиме *OFB*.
10. Докажите, что исходный текст, используемый Алисой, может быть восстановлен Бобом в режиме *CTR*.
11. Покажите диаграмму для шифрования и дешифрования в режиме *CFB*, когда `r = n`.
12. Покажите диаграмму для шифрования и дешифрования в режиме *OFB*, когда `r = n`.
13. Покажите процесс, используемый для алгоритма дешифрования в режиме *ECB*, если применяется захват зашифрованного текста (*CTS*).
14. Покажите диаграммы шифрования и дешифрования для режима *ECB* (только последние два блока), когда используется захват зашифрованного текста (*CTS*).
15. Покажите процесс, используемый для алгоритма дешифрования в режиме *CBC*, если применяется захват зашифрованного текста (*CTS*).
16. Покажите шифрование и диаграмму дешифрования для режима *CBC* (только последние два блока), когда используется захват зашифрованного текста (*CTS*).
17. Объясните, почему нет потребности в захвате зашифрованного текста в режимах *CFB*, *OFB* и *CTR* (*CIPHER FEEDBACK*, OUTPUT *FEEDBACK*).
18. Покажите эффект распространения ошибки, когда *ECB* (*ELECTRONIC* CODEBOOK) использует методику *CTS*.
19. Покажите эффект распространения ошибки, когда *CBC* использует методику *CTS*.
20. Режим Формирование цепочки блоков является вариантом, в котором все предыдущие блоки зашифрованного текста перед шифрованием складываются по модулю два с текущим исходным текстом. Выведите рисунок-диаграмму, которая показывает шифрование и дешифрование.
21. Режим размножения Цепочка блоков шифротекста (PCBC) является вариантом *CBC*, в котором перед шифрованием предыдущий блок исходного текста и предыдущий блок

зашифрованного текста складывается по модулю два с текущим блоком исходного текста. Нарисуйте диаграмму, которая показывает шифрование и дешифрование.

22. Режим Цепочка блоков шифротекста с контрольной суммой (CBCC) является вариантом *CBC*, в котором все предыдущие блоки исходного текста перед шифрованием складываются по модулю два с текущим блоком исходного текста. Нарисуйте диаграмму, чтобы показать шифрование и дешифрование и проиллюстрировать процедуру.
23. В *RC4* покажите первые `20` элементов ключевого потока, если ключ засекречивания — `7` байтов со значениями `1, 2, 3, 4, 5, 6` и `7`. Вы можете при желании написать маленькую программу.
24. В *RC4* найдите значение для ключа засекречивания, который не изменяет матрицу состояний после первого и второго шагов инициализации.
25. Алиса обменивается сообщениями с Бобом, используя в *RC4* для засекречивания `16` -байтовый ключ засекречивания. Ключ засекречивания изменяется каждый раз, используя *рекурсивное определение* $K = (K_{i-1}+K_{i-1}) \bmod 2^{128}$. Покажите, сколькими сообщениями они могут обменяться перед тем, как текст начнет повторяться.
26. В A5/1 найдите максимальный период каждого линейного *регистра сдвига*.
27. В A5/1 найдите значение следующих функций. В каждом случае показать, сколько синхронизируется линейных *регистров сдвига*.
    - `Majority (1, 0, 0)`
    - `Majority (0, 1, 1)`
    - `Majority (0, 0, 0)`
    - `Majority (1, 1, 1)`
28. В A5/1 найдите выражение для мажоритарной функции.
29. Напишите алгоритм дешифрования в *псевдокоде* для режима *ECB*.
30. Напишите алгоритм дешифрования в *псевдокоде* для режима *CBC*.
31. Напишите *псевдокод* алгоритма дешифрования для режима *CFB*.
32. Напишите алгоритм дешифрования в *псевдокоде* для режима *OFB*.
33. Напишите алгоритм дешифрования в *псевдокоде* для режима *CTR*.
34. Напишите алгоритм для `shiftleft` -процедуры, используемой в алгоритме 11.4.

35. Напишите алгоритм для `selectleft` -процедуры, используемой в алгоритме 11.4.
36. Напишите алгоритм для процедуры конкатенации, используемой в алгоритме 11.4.

Простые числа

Эта лекция имеет несколько целей: ввести простые числа и их приложения в криптографии, обсудить некоторые алгоритмы проверки простоты чисел и их эффективность, обсудить алгоритмы разложения на множители и их приложения в криптографии, описать китайскую теорему об остатках и ее приложения, ввести квадратичное сравнение, ввести возведение в степень по модулю и логарифмы.

Асимметрично-ключевая криптография, которую мы обсудим в лекциях 14-15, базируется на некоторых положениях теории чисел, включая теории, связанные с простыми числами, разложением на множители *составных объектов* в простые числа, модульном возведение в степень и логарифмах, квадратичных вычетах и китайской теореме об остатках. Эти проблемы будут рассмотрены здесь, в чтобы упростить понимание лекциии 14-15.

12.1. Простые числа

Асимметрично-ключевая криптография широко использует простые числа. Тема простых чисел — большая часть любой книги по теории чисел. Эта лекция обсуждает только несколько понятий и фактов, чтобы открыть путь к лекциях 14-15.

Определение

Положительные целые числа могут быть разделены на три группы: число `1`, простые числа и *составные объекты*, как это показано на рис. 12.1.

Рис. 12.1. Три группы положительных целых чисел

Положительное целое число — простое число тогда и только тогда, когда оно точно *делимо* без остатка на два целых числа — на 1 и на само себя. Составной объект — положительное целое число больше с чем двумя делителями.

Простое число делимо без остатка только на себя и 1.

Пример 12.1

Какое наименьшее простое число?

Решение

Наименьшее простое число — 2, оно делится без остатка на 2 (само на себя) и 1. Обратите внимание, что целое число 1 — не простое число согласно определению, потому что простое число должно быть *делимо* без остатка двумя различными целыми числами, не больше и не меньше. Целое число 1 *делимо* без остатка только на себя; поэтому 1 — это не простое число.

Пример 12.2

Перечислите простые числа, меньшие, чем 10.

Решение

Есть четыре простых числа меньше чем 10: 2, 3 5 и 7. Интересно, что процент простых чисел в диапазоне 1-10 — 40%. С увеличением

диапазона процент уменьшается.

Взаимно простые числа

Два положительных целых числа `a` и `b` являются взаимно простыми (coprime), если `НОД (a, b) = 1`, потому что число `1` является взаимно простым с любым целым числом. Если `p` — простое число, тогда все числа от `1` до `p–1` являются взаимно простыми к `p`. В лекции 2 мы обсуждали множество Z_n^*, чьи элементы — все числа, взаимно простые с `n`. Множество Z^* является тем же самым, за исключением того, что модуль `(p)` — простое число.

Количество простых чисел

После того как понятие простых чисел было определено, естественно возникает вопрос: число простых чисел конечно или бесконечно? Возьмем число `n`. Сколько есть простых чисел меньших, чем это число, или равных `n`?

Число простых чисел

Число простых чисел бесконечно. Приведем нестрогое доказательство: предположим, что множество простых чисел конечно (ограничено), и пусть `p` — наибольшее простое число. Перемножим все простые числа, входящие в это множество, и получим результат $P = 2 \times 3 \times \cdots \times p$. Целое число `(P + 1)` не может иметь простого делителя $q \leqslant p$ (`p` – наибольшее простое число). Тогда этот делитель должен быть одним из множителей, входящих в `P`. Это значит, что `q` делит `P`. Если `q` также делит `(P + 1)`, то `q` делит `(P + 1) – P = 1`. Единственное число, которое делит `1`, — это сама `1`, которая не является простым числом. Поэтому `q` должно быть большим, чем `p`, и ряд простых чисел не исчерпывается принятым конечным множеством.

Число простых чисел бесконечно.

Пример 12.3

Как тривиальный пример, предположим, что единственные простые числа находятся в множестве {2, 3, 5, 7, 11, 13, 17}. Здесь P = 510510 и P + 1 = 510511. Однако 510511 состоит из следующих простых чисел $510511 = 19 \times 97 \times 277$; ни одно из этих простых чисел не было в первоначальном списке. Эти три простых числа больше, чем 17.

Число простых чисел, меньших n

Чтобы рассмотреть вторую возможность, введем функцию $\pi(n)$, которая определяет число простых чисел, меньших или равных n. Ниже показаны значения этой функции для различного $\pi(n)$.

$$\pi(1) = 0 \quad \pi(2) = 1 \quad \pi(3) = 2 \quad \pi(10) = 4 \quad \pi(20) = 8$$
$$\pi(50) = 15 \quad \pi(100) = 25$$

Но если n является очень большим, как мы можем вычислить $\pi(n)$? Для ответа мы можем использовать только приближение, которое показано ниже:

$$[n/(\ln n)] < \pi(n) < [n/(\ln n - 1.08366)]$$

Гаусс обнаружил верхний предел; Лагранж обнаружил нижний предел.

Пример 12.4

Найдите количество простых чисел, меньших, чем 1 000 000.

Решение

Приближение дает диапазон от 72 383 до 78 543. Фактическое число простых чисел — 78 498.

Проверка на простое число

Следующий вопрос, который приходит на ум: как мы можем определить для данного числа n, является ли оно простым числом? Мы должны проверить, *делимо* ли без остатка это число всеми простыми числами, меньшими, чем $\sqrt{n}$. Мы знаем, что этот метод неэффективен, но он хорош для начала.

Пример 12.5

Действительно ли 97 — простое число?

Решение

Наибольшее ближайшее целое число — $\sqrt{n} = 9$. Простые числа меньше чем 9 — 2, 3, 5 и 7. Проверим, *делимо* ли без остатка 97 любым из этих номеров. Ответ: не *делимо*, так что 97 — простое число.

Пример 12.6

Действительно ли 301 — простое число?

Решение

Наибольшее ближайшее целое число $\sqrt{301} = 17$. Мы должны проверить 2, 3, 5, 7, 11, 13 и 17. Числа 2, 3 и 5 не делят 301, но 7 — делит. Поэтому 301 — не простое число.

Решето Эратосфена

Греческий математик Эратосфен изобрел метод, как найти все простые числа, меньшие, чем n.

Метод назван решетом Эратосфена. Предположим, что мы хотим найти все числа, меньшие, чем 100. Мы записываем все числа между 2 и 100. Поскольку $\sqrt{100} = 10$, мы должны видеть, делим ли без остатка любой номер меньше чем 100 на числа 2, 3, 5 и 7. Таблица 12.1 показывает результат.

Таблица 12.1. *Решето*

Эратосфена

	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100

Процесс состоит в следующем:

1. Вычеркнуть все числа, *делимые* без остатка на `2` (кроме самого `2`).
2. Вычеркнуть все числа, *делимые* без остатка на `3` (кроме самого `3`).
3. Вычеркнуть все числа, *делимые* без остатка на `5` (кроме самого `5`).
4. Вычеркнуть все числа, *делимые* без остатка на `7` (кроме самого `7`).
5. Оставшиеся числа – простые.

Phi-функция Эйлера

Phi-функция Эйлера, $\varphi(n)$, которую иногда называют тотиентой Эйлера, играет очень важную роль в криптографии. Функция $\varphi(n)$ находит из ряда чисел `0, 1…., n-1` числа, взаимно простые с `n`. Можно вспомнить из лекции 2, что множество Z_{n^*} — числа, которые не больше чем `n` и взаимно простые с `n`. Функция $\varphi(n)$ вычисляет число элементов этого множества. Ниже показано, как найти это значение:

1. $\varphi(1) = 0$.
2. $\varphi(p) = p - 1$, если `p` — простое число.

3. $\varphi(m \times n) = \varphi(m) \times \varphi(n)$, если m и n — взаимно простые.
4. $\varphi(p^e) = p^e - p^{e-1}$, если p — простое.

Мы можем объединить эти четыре правила, предназначенные для нахождения $\varphi(n)$.

$$\varphi(n) = ({p_1}^{e_1} - {p_1}^{e_1-1}) \times ({p_2}^{e_2} - {p_2}^{e_2-1}) \times \cdots \times (p^{e_k} - p^{e_k-1})$$

Очень важно заметить, что значение $\varphi(n)$ для больших чисел может быть найдено, если может быть найдено число n и если n может быть представлено в виде разложения простых чисел. Другими словами, трудность нахождения $\varphi(n)$ зависит от трудности нахождения разложения n. Это рассматривается в следующем разделе.

Трудность нахождения $\phi(n)$ зависит от трудности нахождения разложения n.

Пример 12.7

Какое значение имеет $\varphi(13)$?

Решение

Поскольку 13 — простое число, $\varphi(13) = (13 - 1) = 12$.

Пример 12.8

Какое значение имеет $\varphi(10)$?

Решение

Мы можем использовать третье правило: $\varphi(10) = \varphi(2) \times \varphi(5) = 1 \times 4 = 4$, поскольку 2 и 5 — простые числа.

Пример 12.9

Какое значение имеет $\varphi(240)$?

Решение

Мы можем записать $240 = 2^4 \times 3^1 \times 5^1$.

Тогда

$$\varphi(240) = (2^4 - 2^3) \times (3^1 - 3^0) \times (5^1 - 5^0) = 64$$

Пример 12.10

Можно ли утверждать, что $\varphi(49) = \varphi(7) \times \varphi(7) = 6 \times 6 = 36$?

Решение

Нет.

$$\varphi(49) = 7^2 - 7^1 = 42$$

Пример 12.11

Какие числа являются элементами в Z_{14}^*?

Решение

$\varphi(14) = \varphi(7) \times \varphi(2) = 6 \times 1 = 6$. Элементы – это 1, 3, 5, 9, 11 и 13.

Интересный факт: если n > 2, значение $\phi(n)$ — четное.

Малая теорема Ферма

Малая теорема Ферма играет очень важную роль в теории чисел и криптографии. Ниже мы приводим две версии теоремы.

Первая версия

Первая версия говорит, что если `p` — простое число и `a` — целое число, такое, что `p` не является делителем `a`, тогда $a^{p-1} \equiv 1 \mod p$.

Вторая версия

Вторая версия вводит ограничивающие условие на `a`. Она утверждает, что если `p` — простое число и `a` — целое число, то $a^p \equiv a \mod p$.

Приложения

Хотя мы будем рассматривать приложения этой теоремы позже в этой лекции, теорема очень полезна для того, чтобы решить некоторые проблемы.

Возведение в степень. Малая теорема Ферма иногда полезна для того, чтобы быстро найти решение при возведении в степень. Следующие примеры показывают это.

Пример 12.12

Найдите результат $6^{10} \mod 11$.

Решение

Мы имеем $6^{10} \mod 11 \equiv 1$. Это первая версия малой *теоремы Ферма*, где `p = 11`.

Пример 12.13

Найдите результат $3^{12} \mod 11$.

Решение

Здесь степень `(12)` и модуль `(11)` не соответствуют условиям *теоремы Ферма*. Но, применяя преобразования, мы можем привести решение к использованию малой *теоремы Ферма*.

$$3^{12} \bmod 11 \equiv (3^{11} \times 3) \bmod 11 \equiv (3^{11} \bmod 11)(3 \bmod 11) \equiv (3 \times 3) \bmod 11 = 9$$

Мультипликативные инверсии. Очень интересное приложение теорема Ферма находит для некоторых мультипликативных *инверсий*, если модуль — простое число. Если p — простое число и a — целое число, такое, что p не является его делителем, тогда $a^{-1} \bmod p = a^{p-2} \bmod p$. Это может быть легко доказано, если мы умножим обе стороны равенства на a и используем первую версию малой *теоремы Ферма*:

$$a \times a^{-1} \mod p \equiv a \times a^{p-2} \mod p \equiv a^{p-1} \mod p \equiv 1 \mod p$$

Это приложение позволяет не использовать расширенный *алгоритм Евклида* для нахождения мультипликативных *инверсий*.

Пример 12.14

Инверсии по модулю простого числа могут быть найдены без использования расширенного Евклидова алгоритма:

a. $8^{-1} \bmod 17 = 8^{17-2} \bmod 17 = 8^{15} \bmod 17 = 15 \bmod 17$

b. $5^{-1} \bmod 23 = 5^{23-2} \bmod 23 = 5^{21} \bmod 23 = 14 \bmod 23$

c. $60^{101} \bmod 101 = 60^{101-2} \bmod 101 = 60^{99} \bmod 101 = 32 \bmod 101$

d. $22^{-1} \bmod 211 = 22^{211-2}$ moda $211 = 22^{209} \bmod 211 = 48 \bmod 211$

Теорема Эйлера

Теорему Эйлера можно представить как обобщения малой *теоремы Ферма*. Модуль в *теореме Ферма* — простое число, модуль в теореме Эйлера — целое число. Мы вводим две версии этой теоремы.

Первая версия

Первая версия *теоремы Эйлера* подобна первой версии малой *теоремы Ферма*. Если a и n – взаимно простые, то $a^{\varphi(n)} \equiv 1 \mod n$.

Вторая версия

Вторая версия *теоремы Эйлера* подобна второй версии малой *теоремы Ферма*; она устраняет условие, что n должно быть взаимно простым с a. Если $n = p \times q, a < n$, а k — целое число, то $a^{k \times \phi(n)+1} \equiv a \bmod n$.

Приведем нестрогое доказательство второй версии, основанной на первой версии. Поскольку a < n, то возможны три случая:

1. Если a не кратно ни числу p, ни числу q, то a и n – взаимно простые.

$$\mathbf{a}^{k \times \varphi(n)+1} \mod n = (a^{\varphi(n)})^k \times a \mod n = (1)^k \times a \mod n = \times a \mod n$$

2. Если a — кратное число p, $a = (i \times p)$, но не кратно числу q.

$$a^{\varphi(n)} \mod q = (a^{\varphi(q)} \mod q)^{\varphi(p)} \mod q = 1 \rightarrow a^{\varphi(n)} \mod q = 1$$
$$a^{k \times \varphi(n)} \mod q = (a^{\varphi(n)} \mod q)^k \mod q = 1 \rightarrow a^{k \times \varphi(n)} \mod q = 1$$
$$a^{k \times \varphi(n)} \mod q = 1 \rightarrow a^{k \times \varphi(n)} = 1 + j \times q \text{ (интерпретация сравнения)}$$
$$a^{k \times \varphi(n)+1} = a \times (1 + j \times q) = a + j \times q \times a = a + (i \times j) \times q \times p = a + (i \times j) \times n$$
$$a^{k \times \varphi(n)+1} = a + (i \times j) \times n \rightarrow a^{k \times \varphi(n)+1} = a \mod n \text{ (отношение конгруэнтности)}$$

3. Если a кратно q ($a = i \times q$), но не кратно p, доказательство второго случая то же самое, но p и q меняются местами.

Вторая версия теоремы Эйлера используется в криптографической системе RSA (лекция 10).

Приложения

Хотя мы рассмотрим некоторые приложения *теоремы Эйлера* позже в этой лекции, теорема очень полезна для того, чтобы решать некоторые задачи.

Возведение в степень. *Теорема Эйлера* иногда полезна, чтобы быстро найти решение некоторых задач с возведением в степень. Следующие примеры показывают идею этого процесса.

Пример 12.15

Найдите результат $6^{24} \bmod 35$.

Решение

Мы имеем $6^{24} \bmod 35 = 6^{\varphi(35)} \bmod 35 = 1$

Пример 12.16

Найдите результат $20^{62} \bmod 77$.

Решение

Если введем $k = 1$ согласно второй версии, мы имеем:

$$20^{62} \bmod 77 = (20 \bmod 77) \bmod 77 = (20)(20) \bmod 77 = 15$$

Мультипликативные инверсии. *Теорема Эйлера* может использоваться, чтобы найти мультипликативную *инверсию* по простому модулю. *Теорема Эйлера* может применяться, чтобы найти мультипликативные *инверсии* по составному модулю. Если n и a – взаимно простые, то $a^{-1} \bmod n = a^{\varphi(n)-1} \bmod n$. Это может быть легко доказано умножением обеих сторон равенства на a.

$$a^{-1} \bmod n = a \times a^{\varphi(n)-1} \bmod n = a^{\varphi(n)} \bmod n = 1 \bmod n$$

Пример 12.17

Мультипликативная инверсия по составному модулю может быть найдена без использования расширенного евклидова алгоритма, если

мы знаем разложение на множители *составного объекта*:

a. $8^{-1} \bmod 77 = 8^{\varphi(77)-1} \bmod 77 = 8^{59} \bmod 77 = 29 \bmod 77$

b. $7^{-1} \bmod 15 = 7^{\varphi(15)-1} \bmod 15 = 7^{7} \bmod 15 = 13 \bmod 15$

c.

$$6^{-1} \bmod 187 = 7^{\varphi(187)-1} \bmod 187 = 60^{159} \bmod 187 = 53 \bmod 187$$

d.

$$71^{-1} \bmod 100 = 71^{\varphi(100)-1} \bmod 100 = 71^{39} \bmod 100 = 31 \bmod 100$$

Генерация простых чисел

Два математика, Мерсенна и Ферма, попытались получить формулу, которая могла бы генерировать простые числа.

Простые числа Мерсенны

Мерсенна предложил следующую формулу, которую называют номера Мерсенны. Он предполагал, что формула перечисляет все простые числа.

$M_p = 2^p–1$

Если `p` в приведенной выше формуле — простое число, то, как предполагали, M_p должно быть простым числом. Годы спустя было доказано, что не все числа, полученные по формуле Мерсенны, — простые числа. Ниже приведен список некоторых номеров Мерсенны.

$$M_2 = 2^2 - 1 = 3$$
$$M_3 = 2^3 - 1 = 7$$
$$M_5 = 2^5 - 1 = 31$$
$$M_7 = 2^7 - 1 = 127$$
$$M_{11} = 2^{11} - 1 = 2047 \text{ Непростое число } (2047 = 23 \times 89)$$
$$M_{13} = 2^{13} - 1 = 8191$$
$$M_{17} = 2^{17} - 1 = 131071$$

Оказалось, что M_{11} — не простое число. Однако было найдено, что 41 число по формуле Мерсенны — простые; одно из последних найденных чисел Мерсенны — $M_{124036583}$, наибольшее число содержит 7 253 733 цифр. Поиск продолжается.

Ферма пробовал найти формулу, которая генерирует простые числа. Следующая формула — для чисел Ферма:

Число в формуле $M_p = 2^p - 1$, называемое числом Мерсенны, может быть или не быть простым числом.

Простые числа Ферма

Ферма попытался найти формулу для генерации простых чисел. Он предложил следующую формулу, которая теперь называется формулой Ферма, и проверил номера от F_0 (n=0, 1, …) до F_4, но оказалось, что уже F_4 — не простое число.

$$F_n = 2^{2^n} + 1$$

$F_0 = 3$

$F_1 = 17$

$F_2 = 257$

$F_3 = 65537$

$F_4 = 4294967297 = 641 \times 6700417$. Не простое число

Фактически было доказано, что многие номера до F_{24} — составные числа.

12.2. Испытание простоты чисел

Если формулы получения простых чисел, подобно формулам Ферма или Мерсенна, не гарантируют, что полученные числа — простые, то как мы можем генерировать большие простые числа для криптографии? Мы можем только выбрать случайно большое число и провести испытание, чтобы убедиться, что оно — простое.

Нахождение алгоритма, который правильно и эффективно проверяет очень большое целое число и устанавливает: данное число – простое это число или же составной объект, — всегда было проблемой в теории чисел и, следовательно, в криптографии. Однако, недавние исследования (одно из которых мы обсуждаем в этом разделе) выглядят очень перспективными.

Алгоритмы, которые решают эту проблему, могут быть разделены на две обширные категории — детерминированные алгоритмы и вероятностные алгоритмы. Ниже рассматриваются некоторые представители обеих категорий. Детерминированный алгоритм всегда дает правильный ответ. Вероятностный алгоритм дает правильный ответ в большинстве, но не во всех случаях. Хотя детерминированный алгоритм идеален, он обычно менее эффективен, чем соответствующий вероятностный.

Детерминированные алгоритмы

Детерминированный алгоритм, проверяющий простоту чисел, принимает целое число и выдает на выходе признак: это число — простое число или составной объект. До недавнего времени все детерминированные алгоритмы были неэффективны для нахождения больших простых чисел. Как мы коротко покажем, новые взгляды делают эти алгоритмы более перспективными.

Алгоритм теории делимости

Самое элементарное детерминированное испытание на простоту чисел — испытание на делимость. Мы используем в качестве делителей все числа, меньшие, чем $\sqrt{n}$. Если любое из этих чисел делит n, тогда n — составное. Алгоритм 12.1 показывает проверку на делимость в ее примитивной и очень неэффективной форме.

Алгоритм может быть улучшен, если проверять только нечетные номера. Он может быть улучшен далее, если использовать таблицу простых чисел между 2 и $\sqrt{n}$. Число арифметических операций в алгоритме 12.1 — $\sqrt{n}$. Если мы принимаем, что каждая арифметическая операция использует только операцию на один бит (чисто условное соглашение), тогда сложность разрядной операции алгоритма 12.1 — $\sqrt{2^{n_b}} = 2^{n_b/2}$, где n_b – число битов в n. В больших системах, обозначаемых O, сложность может быть оценена $O(2^n)$: экспоненциально (см. приложение L). Другими словами, алгоритм *делимости* неэффективен, если n_b большое.

Сложность побитного испытания делимостью показательна.

Пример 12.18

Предположим, что n имеет 200 битов. Какое число разрядных операций должен был выполнить алгоритм *делимости*?

Решение

Сложность побитовых операций этого алгоритма — $2^{n_b/2}$. Это означает, что алгоритму необходимо провести 2^{100} *битовых операций*. Если алгоритм имеет скорость 2^{30} операций в секунду, то необходимо 2^{70} секунд для проведения испытаний.

```
Тест на делимость (n)
        //n - число тестов на простоту
{
  r ← 2
  while (r < √n)
    {
    if (r|n) return "a composite"// составное
    r ← r + 1
   }
  return "a prime"//простое
}
```

Пример 12.1.

AKS-алгоритм

В 2002 г. индийские ученые Агравал, Каял и Сахсена (Agrawal, Kayal и Saxena) объявили, что они нашли алгоритм для испытания простоты чисел с полиномиальной сложностью времени разрядных операций $O((\log_2 n_b))$. Алгоритм использует тот факт, что $(x - a)^P \equiv (x^P - a) \bmod p$. Интересно наблюдать, что некоторые будущие разработки делают этот алгоритм стандартным тестом для определения простоты чисел в математике и информатике.

Пример 12.19

Предположим, что n имеет 200 битов. Какое число разрядных операций должен был выполнить алгоритм AKS?

Решение

Сложность разрядной операции этого алгоритма — $O((\log_2 n_b)^{12})$. Это означает, что алгоритму надо только $(\log_2 200)^{12} = 39\,547\,615\,483$ *битовых операций*. На компьютере, способном выполнить 1 миллиард битов в секунду, алгоритму требуется только 40 секунд.

Вероятностные алгоритмы

До AKS-алгоритма все эффективные методы для испытания простоты чисел были вероятностные. Эти методы могут использоваться еще некоторое время, пока AKS формально не принят как стандарт.

Вероятностный алгоритм не гарантирует правильность результата. Однако мы можем получить вероятность ошибки настолько маленькую, что это почти гарантирует, что алгоритм вырабатывает правильный ответ. Сложность разрядной операции алгоритма может стать полиномиальной, при этом мы допускаем небольшой шанс для ошибок. Вероятностный алгоритм в этой категории возвращает результат либо простое число, либо составной объект, основываясь на следующих правилах:

a. Если целое число, которое будет проверено, — фактически простое число, алгоритм явно возвратит простое число.

b. Если целое число, которое будет проверено, — фактически *составной объект*, алгоритм возвращает составной объект с вероятностью $1 - \varepsilon$, но может возвратить простое число с ε вероятности. Вероятность ошибки может быть улучшена, если мы выполняем алгоритм несколько раз с различными параметрами или с использованием различных методов. Если мы выполняем алгоритм m раз, вероятность ошибки может уменьшиться до $_m$.

Тест Ферма

Первый вероятностный метод, который мы обсуждаем, — испытание простоты чисел тестом Ферма.

Если n — простое число, то $a^{n-1} \equiv 1 \mod n$.

Обратите внимание, что если n — простое число, то сравнение справедливо. Это не означает, что если сравнение справедливо, то n — простое число. Целое число может быть простым числом или *составным объектом*. Мы можем определить следующие положения

как тест Ферма:

$$\text{Если n - простое число, то } a^{n-1} \equiv 1 \mod n$$
$$\text{Если n - составной объект, то возможно, что } a^{n-1} \equiv 1 \mod n$$

Простое число удовлетворяет тесту Ферма. *Составной объект* может пройти тест Ферма с вероятностью ε. Сложность разрядной операции испытания Ферма равна сложности алгоритма, который вычисляет возведение в степень. Позже в этой лекции мы приводим алгоритм для быстрого возведения в степень со сложностью разрядной операции $O(n_b)$, где O — номер битов в n. Вероятность может быть улучшена, если проверка делается с несколькими числами (a_1, a_2 и так далее). Каждое испытание увеличивает вероятность, что испытуемое число – это простое число.

Пример 12.20

Проведите испытание Ферма для числа 561.

Решение

Используем в качестве основания число 2.

$$2^{561-1} = 1 \mod 561$$

Число прошло тест Ферма, но это — не простое число, потому что

$$561 = 33 \times 17.$$

Испытание квадратным корнем

В модульной арифметике, если n — простое число, то квадратный корень равен только 1 (либо $+1$, либо -1). Если n — *составной объект*, то квадратный корень — $+1$ или (-1), но могут быть и другие корни. Это называют испытанием простоты чисел квадратным корнем. Обратите внимание, что в модульной арифметике -1 означает $n-1$.

Если n - простое число, $\sqrt{1} \bmod n = \pm 1$
Если n - составной объект, $\sqrt{1} \bmod n = \pm 1$, и возможны другие значения

Пример 12.21

Каковы квадратные корни `1 mod n`, если `n` равно `7` (простое число)?

Решение

Единственные квадратные корни `1 mod n` – это числа `1` и `-1`. Мы можем видеть, что

$1^2 = 1 \bmod 7 \quad (-1)^2 = 1 \bmod 7$
$2^2 = 4 \bmod 7 \quad (-2)^2 = 4 \bmod 7$
$3^2 = 2 \bmod 7 \quad (-3)^2 = 2 \bmod 7$

Заметим, что тест не дает результатов для `4, 5` и `6`, потому что `4 = -3 mod 7`, `5 = -2 mod 7` и `6 = -1 mod 7`.

Пример 12.22

Каков квадратный корень из `1 mod n`, если `n` равно `8` (составное)?

Решение

Имеется три решения: `1, 3, 5` и `7` (которые дают `-1`). Мы можем также видеть, что

$1^2 = 1 \bmod 8 \quad (-1)^2 = 1 \bmod 8$
$3^2 = 1 \bmod 8 \quad (-5)^2 = 1 \bmod 8$

Пример 12.23

Каков квадратный корень из `1 mod n`, если `n` равно `17` (простое)?

Решение

Имеются только два решения, соответствующие поставленной задаче: это `1` и `(-1)`.

$1^2 = 1 \bmod 17 \quad (-1)^2 = 1 \bmod 17$
$2^2 = 4 \bmod 17 \quad (-2)^2 = 4 \bmod 17$
$3^2 = 9 \bmod 17 \quad (-3)^2 = 9 \bmod 17$
$4^2 = 16 \bmod 17 \quad (-4)^2 = 16 \bmod 17$
$5^2 = 8 \bmod 17 \quad (-5)^2 = 8 \bmod 17$
$6^2 = 2 \bmod 17 \quad (-6)^2 = 2 \bmod 17$
$7^2 = 15 \bmod 17 \quad (-7)^2 = 15 \bmod 17$
$8^2 = 13 \bmod 17 \quad (-8)^2 = 13 \bmod 17$

Заметим, что не надо проверять целые числа, большие 8, потому что $9 = -8 \bmod 17$

Пример 12.24

Каков квадратный корень из $1 \bmod n$, если n равно 22 (составное)?

Решение

Сюрприз в том, что имеется только два решения: $+1$ и -1, хотя 22 — составное число.

$1^2 = 1 \bmod 22$
$(-1)^2 = 1 \bmod 22$

Хотя во многих случаях имеется испытание, которое показывает нам однозначно, что число составное, но это испытание провести трудно. Когда дано число n, то все числа, меньшие, чем n (кроме чисел 1 и $n-1$), должны быть возведены в квадрат, чтобы гарантировать, что ни одно из них не равно 1. Это испытание может использоваться для чисел (не $+1$ или -1), которые в квадрате по модулю n дают значение 1. Этот факт помогает в испытании Миллера–Рабина, которое рассматривается в следующем разделе.

Тест Миллера-Рабина

Тест Миллера-Рабина определения простого числа есть комбинация тестов Ферма и квадратного корня. Он элегантным способом находит

сильное псевдопростое число (простое число с очень высокой вероятностью). В этом тесте мы записываем `n-1` как произведение нечетного числа `m` и степени числа `2`.

$$n-1 = m \times 2^k$$

В тесте Ферма при основании `a` можно записать так, как это показано ниже.

Идея теста на простоту числа на основе Ферма

$$a^{n-1} = a^{m \times 2k} = [a^m]^{2k} = [a^m]^2$$

Другими словами, вместо того чтобы вычислять $a^{n-1} \pmod n$ в один шаг, мы можем сделать это в `k + 1` шагов. Какое преимущество в таком применении? Преимущество заключается именно в том, что испытание квадратным корнем может быть выполнено на каждом шаге. Если квадратный корень показывает сомнительные результаты, мы останавливаемся и объявляем `n` составным номером. На каждом шаге мы обеспечиваем, что тест Ферма и испытание квадратным корнем удовлетворено на всех парах смежных шагов, если оно удовлетворительно (если результат равен `1`).

Инициализация

Выберите основу и вычислите $T = a^m$, в который $m = (n - 1) / 2^k$.

а. Если `T` равно `+1` или `-1`, объявляют, что `n` — сильное псевдопростое число, и процесс останавливается. Мы говорим, что `n` прошел два испытания: тест Ферма и испытание квадратным корнем. Почему? Потому что если `T` равно ± 1, то `T` станет `1` на следующем шаге и остается `1` до прохождения теста Ферма. Кроме того, `T` прошел испытание тестом квадратного корня, потому что `T` был бы равен `1` на следующем шаге и квадратный корень был бы равен `1` (на следующем шаге) и равен ± 1 (на этом шаге).

b. Если T равен другому значению, мы не уверены, является ли n простым числом или *составным объектом*, значит, процесс будет продолжаться на следующем шаге.

Шаг 1

Возводим T в квадрат.

a. Если результат равен +1, мы определенно знаем, что тест Ферма пройден, потому что T остается 1 для последующих испытаний. Испытание квадратным корнем, однако, не прошло. Поскольку T равно 1 на этом шаге и имело на предыдущем шаге другое значение, чем ± 1 (причина, почему мы не остановились на предыдущем шаге), n объявляют *составным объектом*, и процесс останавливается.

b. Если результат равен (−1), мы знаем, что n в конечном счете пройдет тест Ферма. Мы знаем, что он пройдет испытание квадратным корнем, потому что T равно (−1) в этом шаге и станет 1 на следующем шаге. Мы объявляем n сильным псевдослучайным простым числом и останавливаем процесс.

c. Если T имеет еще какое-либо значение, мы не уверены, имеем ли мы дело с простым числом, и процесс продолжается на следующем шаге.

Шаги 2 до k–1

Этот шаг и все остальные шаги до k−1 такие же, как и шаг 1.

Этот шаг не является необходимым. Если мы достигли его и не приняли решение, он не поможет нам. Если результат этого шага (−1), значит, тест Ферма пройден, но поскольку результат предыдущего шага — не ± 1, испытание квадратное корня не пройдено. После шага k − 1, если процесс не остановлен, мы объявляем, что n — составное.

Тест Миллера-Рабина требует от 0 до k–1.

Алгоритм 12.2 показывает *псевдокод* для теста Миллера-Рабина.

```
Тест Миллера-Рабина (n, a)      // n - число; a - основание
{
  Find m and k such that n–1 = m × 2^k
  T ← a^m mod n
  if ( T = ±1) return "a prime"
  for (I ← 1 to k-1)
// k-1 - максимальное число шагов
 {
   T ← T^2 mod n
   if (T = +1) return "a composite"   // составное
   if (T = -1) return "a prime"    // простое
 }
return "a composite
```

Пример 12.2. Псевдокод для теста Миллера-Рабина

Существует доказательство, что каждый раз, когда для числа проводится тест Миллера-Рабина, вероятность получить результат "не простое число" — 1/4. Если прошло m тестов (с m различными основаниями), вероятность, что тест выдаст не простое число — $(1/4)^m$.

Пример 12.25

Проведите тест Миллера-Рабина к числу 561.

Решение

Используя основание 2, получим $561 - 1 = 35 \times 2^4$, что означает, что m = 35, k = 4 и a = 2

Инициализация: $T = 2^{35} \bmod 561 = 263 \bmod 561$

k = 1 $T = 263^2 \bmod 561 = 166 \bmod 561$

k = 2 $T = 166^2 \bmod 561 = 67 \bmod 561$

k = 3 $T = 67^2 \bmod 561 = +1 \bmod 561 \rightarrow$ составное

Пример 12.26

Мы уже знаем, что 27 — не простое число. Попробуем применить тест

Миллера-Рабина.

Решение

Основание равно 2, тогда $27 - 1 = 13 \times 2^1$, что означает m = 13, k = 1 и a = 2. В этом случае k - 1 = 0, и мы должны сделать только шаг инициализации:

$T = 2^{13} \bmod 27 = 11 \bmod 27$. Однако поскольку алгоритм не делает ни одного цикла, вырабатывается решение "составной объект".

Пример 12.27

Мы знаем, что 61 — простое число; давайте посмотрим, что даст тест Миллера-Рабина

Решение

Мы используем основание 2.

$$61\text{–}1 = 15 \times 2^2 \rightarrow m = 15k = 2a = 2$$

Инициализация: $T = 2^{15} \bmod 61 = 11 \bmod 61$

к = 1 $T = 11^2 \bmod 61 = -1 \bmod 61 \rightarrow$ простое число

Обратите внимание, что последний результат — это 60 mod 61, но мы знаем, что 60 = -1 mod 61.

Рекомендованные тесты простоты чисел

Сегодня один из самых популярных тестов простоты чисел — комбинация теории *делимости* и тест Миллера-Рабина. При этом рекомендуются следующее шаги:

1. Выбрать нечетное целое число, потому что все четные целые числа (кроме 2) — явно *составные объекты*.
2. Сделать некоторые тривиальные испытания теории *делимости* на некоторых известных простых числах, таких как 3, 5, 7, 11, 13: так, чтобы убедиться, что вы не имеете дело с очевидным

составным объектом. Если они не являются делителями при всех этих испытаниях, сделайте следующий шаг. Если выбранное число не прошло хотя бы один из этих тестов, вернитесь на один шаг и выберите другое нечетное число.

3. Выбрать набор оснований для теста. Большое множество оснований предпочтительно.
4. Сделать тест Миллера-Рабина на каждом из оснований. Если любой из них не проходит, вернитесь на один шаг и выберите другой нечетный номер. Если тесты прошли для всех оснований, объявите это число как сильное псевдопростое число.

Пример 12.28

Номер `4033` — *составной объект* (37×109). Это подтверждает рекомендованное испытание простоты чисел?

Решение

1. Выполним проверку согласно теории *делимости*. Проверим сначала числа `2, 3, 5, 7, 11, 17` и `23` – не являются делителями числа `4033`.

2. Выполним испытание Миллера-Рабина с основанием `2`, тогда $4033 - 1 = 63 \times 2^6$, что означает `m = 63` и `k = 6`.

Инициализация: $T \equiv 2^{63} (mod\ 4033) \equiv 3521 (mod\ 4033)$

$k = 1 \qquad T \equiv T^2 = 3521^2 (mod\ 4033) = -1 (mod\ 4033) \rightarrow$ Тест прошел

3. Но мы не удовлетворены. Мы продолжаем с другим основанием — `3`.

Инициализация: $T \equiv 3^{63} (mod\ 4033) \equiv 3551 (mod\ 4033)$

$k = 1 \qquad T \equiv T^2 \equiv 3551^2 (mod\ 4033 \equiv 2443 (mod\ 4033)$

$k = 2 \qquad T \equiv T^2 \equiv 2443^2 (mod\ 4033 \equiv 3442 (mod\ 4033)$

$k = 3 \qquad T \equiv T^2 \equiv 3442^2 (mod\ 4033 \equiv 2443 (mod\ 4033)$

$k = 4 \qquad T \equiv T^2 \equiv 2443^2 (mod\ 4033 \equiv 3442 (mod\ 4033)$

$k = 5 \qquad T \equiv T^2 \equiv 3442^2 (mod\ 4033 \equiv 2443 (mod\ 4033)$ Не соответствует (составное)

12.3. Разложение на множители

Разложение на множители — предмет непрерывного исследования в прошлом; и такие же исследования, вероятно, продолжатся в будущем. Разложение на множители играет очень важную роль в безопасности некоторых криптосистем с открытым ключом (см. лекции 14-15).

Основная теорема арифметики

Согласно Основной теореме арифметики любое положительное целое число больше единицы может быть уникально записано в следующей главной форме разложения на множители, где p_1, p_2, ..., p_k — простые числа и e_1, e_2, ..., e_k — положительные целые числа.

$$n = p_1^{e1} \times p_2^{e2} \times \cdots \times p_{e_k}^{k}$$

Есть непосредственные приложения разложения на множители, такие как вычисление наибольшего общего делителя и наименьшего общего множителя.

Наибольший общий делитель

В лекции 2 мы уже обсуждали наибольший общий делитель двух номеров, НОД (a, b). Посмотрите, как евклидов алгоритм дает это значение, но это значение может также быть найдено, если мы знаем разложение на множители чисел a и b.

$$a = {p_1}^{a_1} \times {p_2}^{a_2} \times \ldots \times {p_k}^{a_k}$$
$$b = {p_1}^{b_1} \times {p_2}^{b_2} \times \ldots \times {p_k}^{b_k}$$
$$HOD(a,b) = {p_1}^{\min(a_1,b_1)} \times {p_2}^{\min(a_2,b_1)} \times \ldots \times {p_k}^{\min(a_k,b_1)}$$

Наименьшее общее кратное

Наименьшее общее кратное, НОК (a, b), — наименьшее целое

число, кратное числам a и b. Используя разложение, мы также находим НОК (a, b).

$$a = {p_1}^{a_1} \times {p_2}^{a_2} \times \ldots \times {p_k}^{a_k}$$
$$HOD(a,b) = {p_1}^{\max(a_1,b_1)} \times {p_2}^{\max(a_2,b_1)} \times \ldots \times {p_k}^{\max(a_k,b_1)}$$

Может быть доказано, что НОД (a,b) и НОК (a,b) связаны с друг другом, как это показано ниже:

$$\text{НОК}(a,b) \times \text{НОД}(a,b) = a \times b$$

Методы разложения на множители

Поиск эффективных алгоритмов для разложения на множители больших составных чисел ведется давно. К сожалению, совершенный алгоритм для этого пока не найден. Хотя есть несколько алгоритмов, которые могут разложить число на множители, ни один не способен провести разложение достаточно больших чисел в разумное время. Позже мы увидим, что это хорошо для криптографии, потому что современные *криптографические системы* полагаются на этот факт. В этой секции мы даем несколько простых алгоритмов, которые проводят разложение составного числа. Цель состоит в том, чтобы сделать процесс разложения на множители менее трудоёмким.

Метод проверки делением

Самый простой и наименее эффективный алгоритм — метод разложения на множители проверкой делением. Мы просто пробуем все положительные целые числа начиная с 2, для того чтобы найти одно, которое делит n. После обсуждения решета Эратосфена мы знаем, что если n составное, то делитель будет простым числом $p \leqslant \sqrt{n}$. Алгоритм 12.3 показывает этот метод. Алгоритм имеет два цикла: один внешний и один внутренний, находит уникальные множители в разложении; внутренняя петля находит повторяющиеся множители разложения. Например, $24 = 2^3 \times 3$. Внешний цикл множители 2 и 3. Внутренний цикл находит, что число 2 множитель.

```
разложение проверкой_ делением (n)
{ // n раскладываемое число
  a ← 2
  while (a ≤ √n)
  {
   while (n mod a = 0)
   {
     output a     // элементы выхода "один за другим"
     n = n/a
    }
   a ← a + 1
   }
 if (n > 1) output n     // n не имеет больше множителей
}
```

Пример 12.3. Псевдокод для разложения на множители для метода проверки делением

Сложность. Метод проверки делением обычно хорош, если $n < 2^{10}$, но он неэффективен и неосуществим для разложения больших целых чисел. Сложность алгоритма (приложение L) показательна.

Пример 12.29

Используйте алгоритм проверки делением, чтобы найти сомножители числа 1233.

Решение

Мы выполняем программу, основанную на алгоритме, и получаем следующий результат:

$$1233 = 3^2 \times 137$$

Пример 12.30

Используйте алгоритм проверки делением, чтобы найти сомножители 1523357784.

Решение

Мы выполняем программу, основанную на алгоритме, и получаем следующий результат:

$$1523357784 = 2^3 \times 3^2 \times 13 \times 37 \times 43987$$

Метод Ферма

Метод Ферма разложения на множители (алгоритм 12.4) делит номер n на два положительных целых числа (a и b — не обязательно простые числа) так, чтобы $n = a \times b$.

```
Разложение_ на_ множители Ферма (n)     // n - раскладываемое число
{
  x ← √n
  while (< n) // наименьшее целое, большее, чем √n
    {

    w ← x²–n
    if (w полный квадрат числа) y ← √w;  a ← x+y;  b ← x–y;  return
a and b
    x ← x + 1
    }
}
```

Пример 12.4. Алгоритм 12.4. Псевдокод для разложения на множители по методу Ферма

Метод Ферма основан на факте, что если мы можем найти x и y, такие, что n = x^2 – y^2, тогда мы имеем

$$n = x^2 - y^2 = a \times b \text{ при } a = (x + y) \text{ и } b = (x - y)$$

Метод сводится к попытке найти два целых числа a и b, близкие друг к другу ($a \approx b$). Начинаем с наименьшего целого числа, большего, чем

$x = \sqrt{n}$. Потом пробуем найти другое целое число y, такое, чтобы выполнялось уравнение $y^2 = x^2 - n$. В каждой итерации мы должны рассмотреть, является ли результат $x^2 - n$ полным квадратом. Если мы находим такое значение для y, мы вычисляем a и b и выходим из цикла. Если мы не делаем этого, мы проводим другую итерацию.

Заметим, что метод не обязательно находит разложение на простые числа (*каноническое разложение*); алгоритм должен быть повторен рекурсивно для каждого из значений a и b, пока не будут найдены сомножители в виде простых чисел.

Сложность. Сложность метода Ферма является близкой к показательному закону (см. приложение L).

p – 1 Метод Полларда

В 1974 г. Джон Поллард разработал метод, который находит разложение числа p на простые числа. Метод основан на условии, что $p - 1$ не имеет сомножителя, большего, чем заранее определенное значение B, называемое границей. Алгоритм Полларда показывает, что в этом случае

$$p = НОД\ (2^{B!} - 1, n)$$

Алгоритм 12.5 показывает *псевдокод* для p – 1 метода Полларда разложения на множители. Когда мы выходим из второго цикла, в a сохраняется $2^{B!}$.

```
Pollard_ (p-1)_ Factorization (n,B)
{       // n - раскладываемое число
  a ← 2
  e ← 2
  while (e ≤ B)
    {
    a ← a^e  mod  n
    e ← e + 1
   }
  p ←  gsd (a–1, n)    // gsd - НОД (наибольший общий делитель)
  if 1 < p < n return p
  return failure
}
```

Пример 12.5. Псевдокод для p – 1 метода Полларда разложения на множители

Сложность. Заметим, что этот метод требует сделать `B - 1` операций возведения в степень `(a = a e mod n)`. Как мы увидим позже в этой лекции, есть быстрый алгоритм возведения в степень, который выполняет это за `2 log2 B` операций. Метод также использует вычисления `НОД`, который требует n^3 операций. Мы можем сказать, что сложность — так или иначе больше, чем `O(B)` или $O(2^{n_b})$, где n_b — число битов в `B`. Другая проблема – этот алгоритм может заканчиваться сигналом об ошибке. Вероятность успеха очень мала, если `B` имеет значение, не очень близкое к величине $\sqrt{n}$.

Пример 12.31

Используя `p - 1` метод Полларда, найдите сомножители числа `57247159` с границей `B = 8`.

Решение

Мы выполняем программу, основанную на рассмотренном выше алгоритме, и находим, что `p = 421`. Фактически $57247159 = 421 \times 135979$. Обратите внимание, что `421` — простое

число и p −1 не имеет ни одного сомножителя, большего 8, т.е. ($421 - 1 = 2^2 \times 3 \times 5 \times 7$).

PO (Rho) – метод Полларда

В 1975 г. Джон М. Поллард разработал второй метод для разложения на множители, который базируется на следующих положениях:

a. Предположим, что есть два целых числа, x_1 и x_2, таких, что p делит $x_1 - x_2$, но эта разность не делится на n.

b. Может быть доказано, что p = НОД $(x_1 - x_2, n)$. Поскольку p делит $x_1 - x_2$, можно записать, что $x_1 - x_2 = q \times p$. Но поскольку n не делит $x_1 - x_2$, очевидно, что q не делится n. Это означает, что НОД $(x_1 - x_2, n)$ является либо 1, либо сомножитель.

Следующий алгоритм повторно выбирает x_1 и x_2, пока не находит соответствующую пару.

1. Выберите x_1 — малое случайное целое число, называемое первоисточником.
2. Используйте функцию, чтобы вычислить x_2, такую, чтобы n не делило $x_1 - x_2$. Функция, которая может быть применена, — это $x_2 = f(x_1) = x_1^2 + a$ (a обычно выбирается как 1).
3. Вычислить НОД $(x_1 - x_2, n)$. Если это не 1, результат – сомножитель. Алгоритм останавливается. Если это 1, то происходит возвращение, чтобы повторить процесс с x_1. Теперь мы вычисляем x_3. Заметим, что в следующем раунде мы начинаем с x_3 и так далее. Если мы перечислим значения нескольких x, используя PO (rho) алгоритм Полларда, мы увидим, что дуга значений в конечном счете повторяется, создавая форму, подобную греческой букве PO (rho) или в греческом алфавите ρ), как это показано на рис. 12.2.

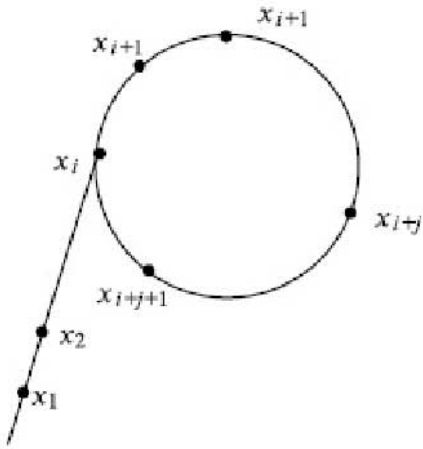


Рис. 12.2. Успешные числа в Ро алгоритме Полларда

Чтобы уменьшить число итераций, алгоритм был немного изменен. Он начинается с пары (x_0, x_0), и *итеративно* вычисляет (x_1, x_2), (x_2, x_4), (x_3, x_6), …. (x_i, x_{2i}), используя равенство $x_{i+1} = f(x_i)$. В каждой итерации мы применяем функцию $f(x_i)$ (начиная с шага 2). При этом вычисление идут следующим образом: в паре вычисляется один раз первый элемент и дважды вычисляется второй элемент (см. алгоритм 12.6).

```
Pollard_ rho _ Factorization (n, B)    // n - число, которое надо разложить
{
  x ← 2
  y ← 2
  p ← i
  while (p = 1)
   {
    x ← f(x) mod n
    y ← f(f(y) mod n) mod n
    p ← gcd(x − y, n)     // gsd (a,b) - это НОД (a,b)
   }
  return p
}     // если p = n, программа не выполнена
```

Пример 12.6. Алгоритм 12.6

Сложность. Метод требует $\sqrt{p}$ арифметических операций. Однако поскольку мы предполагаем, что p будет меньше или равняться $\sqrt{n}$, мы ожидаем около $n^{1/4}$ арифметические операций. Это означает, что сложность разрядной операции 0 ($2^{n_b/4}$) показательна.

Пример 12.32

Предположим, что есть компьютер, который, может выполнить 2^{30} (почти 1 миллиард) разрядных операций в секунду. Какое приблизительно время потребуется, чтобы разложить на множители целое число размера

a. 60 десятичных цифр

b. 100 десятичных цифр

Решение

a. Множество 60 десятичных цифр имеют почти 200 битов. Сложность — $2^{n_b/4}$или 2^{50}. Со скоростью 2^{30} операций в секунду алгоритм может быть выполнен в 2^{20} секунды или почти за 12 дней

b. Множество 100 десятичных цифр — это почти 300 битов. Сложность — 2^{75}. Со скоростью 2^{30} операций в секунду алгоритм может быть выполнен в 2^{45} секунд или за много лет

Пример 12.33

Мы написали программу, чтобы вычислить разложение 434617. Результат — 709 ($434617 = 709 \times 613$) Таблица 12.2 показывает значения пар (x и y) и p в этом процессе.

Более эффективные методы

В течение прошлых нескольких десятилетий были изобретены несколько методов разложения на множители, они кратко рассматриваются ниже.

Таблица 12.2. Значения x, y и p в примере 12.33

x	y	p
2	2	1
5	26	1
26	23713	1
677	142292	1
23713	157099	1
345589	52128	1
142292	41831	1
380320	68775	1
157099	427553	1
369457	2634	1
52128	63593	1
102901	161353	1
41831	64890	1
64520	21979	1
68775	16309	709

Квадратичное решето

Померанс изобрел метод разложения на множители, называемый методом квадратичного решета. Метод применяет процедуру просеивания, чтобы найти значение $x^2 \bmod n$. Метод используется, чтобы разложить на множители целые числа с более чем 100 цифрами.

Его сложность — $0\ (e^c)$, где $C \approx 2(\ln n \ln \ln n)^{1/2}$. Обратите внимание, что это – субпотенциальная сложность.

Решето поля чисел

Эндрик Ленстра и Арджин Ленстра изобрели метод разложения на множители и назвали его метод решета поля чисел. Метод использует процедуру просеивания в алгебраической кольцевой структуре к $x^2 \equiv y^2 \bmod n$. Показано, что этот метод быстрее для разложения чисел с более чем 120 десятичными цифрами. Его сложность – $O(e^C)$ где $C \approx (\ln n)^{1/3}(\ln \ln n)^{2/3}$. Обратите внимание, что это — также субпоказательная сложность.

Пример 12.34

Предположим, что есть компьютер, который может выполнить 2^{30} (почти 1 миллиард) *битовых операций* в секунду. Какое приблизительно время требуется для этого компьютера, чтобы разложить на множители целое число из 100 десятичных цифр, используя один из следующих методов?

a. Метод квадратичного решета

b. Метод решета поля чисел

Решение

Номер с 100 десятичными цифрами имеет почти 300 битов ($n = 2^{300}$).

$\ln (2^{300}) = 207$ и $\ln\ln (2^{300}) = 5$.

a. Для метода квадратичного решета мы имеем

$$(207)^{1/2} \times (5)^{1/2} = 14 \times 2,23 = 32.$$

Это означает, что нам надо e^{32} *битовых операций*, которые могут быть сделаны в $(e^{32}) / (2^{30}) = 20$ часов.

b. При методе решета поля чисел мы имеем

$(207) \times (5)^{2/2} = 6 \times 3 \approx 18$. Это означает, что нам надо e^{18} *битовых операций*, которые могут быть сделаны за $(e^{30})/(2^{30}) \approx 6$ секунд. Однако эти результаты правильны, только если мы имеем компьютер, который может выполнить 1 миллиард *битовых операций* в секунду.

Другие проблемы

В лекциях 14-15 мы обсудим прикладные вопросы задачи разложения на множители для вскрытия криптосистем с открытым ключом. Если будут изобретены более эффективные методы разложения на множители, то *криптосистемы* с открытым ключом вынуждены будут использовать большие целые числа, чтобы противостоять *криптоанализу*. Изобретатели *RSA* создали основу для конкуренции методов разложения на множители номеров до 2048 битов (больше чем 600 цифр).

12.4. Китайская теорема об остатках

Китайская теорема об остатках (*CRT* — Chinese Reminder Theorem) используется, чтобы решить множество уравнений с одной переменной, но различными взаимно простыми модулями, как это показано ниже:

$$\begin{aligned} \mathrm{x} &\equiv a_1 (mod\ m_1) \\ \mathrm{x} &\equiv a_2 (mod\ m_2) \\ &\ldots \\ \mathrm{x} &\equiv a_k (mod\ m_k) \end{aligned}$$

Китайская теорема об остатках утверждает, что вышеупомянутые уравнения имеют единственное решение, если модули являются взаимно простыми.

Пример 12.35

Следующий пример содержит систему уравнений с различными модулями:

$$x \equiv 2(mod\ 3)$$
$$x \equiv 3(mod\ 5)$$
$$x \equiv 2(mod\ 7)$$

Для этой системы уравнений `x = 23`. Это значение удовлетворяет все уравнения: $23 \equiv 2\,(\mathrm{mod}3)$, $23 \equiv 3\,(\mathrm{mod}5)$, $23 \equiv 2\,(\mathrm{mod}7)$.

Решение

Решение системы уравнений выполняется в следующем порядке:

1. Найти $M = m_1 \times m_2 \times \ldots \times m_k$. Это общий модуль.
2. Найти $M_1 = M/m_1$, $M_2 = M/m_2, \ldots$, $M_k = M/m_k$.
3. Используя соответствующие модули m_1, $m_2, \ldots$, m_k, найти мультипликативную *инверсию* M_1, $M_2, \ldots$, M_k. Обозначим ее M_1^{-1}, $M_2^{-1}, \ldots$, M_k^{-1}.
4. Решение системы уравнений

$$x = (a_1 \times M_1 \times M_1^{-1} + a_2 \times M_2 \times M_2^{-1} + \cdots + a_k \times M_k \times M_k^{-1}) \mod M$$

Обратите внимание, что система уравнений может иметь решение, даже если модули не взаимно простые. Однако в криптографии мы интересуемся только решением уравнений с взаимно простыми модулями.

Пример 12.36

Найдите решение системы уравнений

$$x \equiv 2(mod\ 3)$$
$$x \equiv 3(mod\ 5)$$
$$x \equiv 2(mod\ 7)$$

Из предыдущего примера мы уже знаем, что ответ `x = 23`. Определим его в четыре шага.

Решение

1. $M = 3 \times 5 \times 7 = 105$
2. $M_1 = 105/3 = 35$, $M_2 = 105/5 = 21$, $M_3 = 105/7 = 15$
3. *Инверсии* $M_1^{-1} = 2$, $M_2^{-1} = 1$, $M_3^{-1} = 1$
4. $x = 2 \times 35 \times 2 + 3 \times 21 \times 1 + 2 \times 15 \times 1 = 23 \bmod 105$

Пример 12.37

Найти целое, которое дает в остатке 3, если его разделить на 7 и 13, но без остатка делится на 12.

Решение

Это проблема китайской теоремы об остатке. Мы можем составить три уравнения и найти значение x.

$$x \equiv 3 (mod\ 7)$$
$$x \equiv 3 (mod\ 13)$$
$$x \equiv 0 (mod\ 12)$$

Если проведем четыре шага, мы найдем $x = 276$. Можем проверить, что $276 = 3 \bmod 7$, $276 = 3 \bmod 13$ и 276 делится на 12 (частное 23 и остаток 0).

Приложения

Китайская теорема об остатках часто применяется в криптографии. Одно из таких применений – решение квадратных уравнений — будет обсуждаться в следующей секции. Другое приложение — представление очень большого числа в виде списка малых целых чисел.

Пример 12.38

Предположим, нам надо вычислить $z = x + y$, где $x = 123$ и $y = 334$, но система принимает только числа меньше 100. Эти числа можно представить следующими уравнениями:

$x \equiv 24(mod\ 99)$ $y \equiv 37(mod\ 99)$ $x \equiv 25(mod\ 98)$
$y \equiv 40(mod\ 98)$ $x \equiv 26(mod\ 97)$ $y \equiv 43(mod\ 97)$

Сложим каждое уравнение `x` с соответствующим уравнением `y`:

$$x + y \equiv 61(mod\ 99) \rightarrow z \equiv 61(mod\ 99) x + y \equiv 65(mod\ 98) \rightarrow z \equiv 65(mod\ 98) x + y \equiv 69(mod\ 97) \rightarrow z \equiv 69(mod\ 97)$$

Теперь эти три уравнения могут быть решены, используя китайскую теорему об остатках, чтобы найти `z`. Один из приемлемых ответов равен `z = 457`.

Квадратичное сравнение

Линейное сравнение уже рассматривалось в лекции 2, а китайская теорема об остатках была обсуждена в предыдущей секции. Для решения задач криптографии мы также должны уметь решать квадратичное сравнение, имеющее следующую форму a2x2 + a1x + a0 = 0 (mod n).

Мы ограничим наше обсуждение только квадратичными уравнениями, в которых $a_2 = 1$ и $a_1 = 0$. Тогда рассмотрение будет касаться уравнений следующей формы:

$$x^2 \equiv a \pmod{n}.$$

Квадратичное сравнение с модулем в виде простого числа

Мы сначала рассматриваем случай, в котором модуль является простым числом. Другими словами, мы хотим найти решения уравнения формы $x^2 \equiv \pmod{p}$, в котором p является простым числом и a — целое число, такое, что p и a — взаимно простые. Может быть доказано, что этот тип уравнения либо не имеет никакого решения, либо имеет только два неконгруэнтных решения.

Пример 13.1

Уравнение $x^2 \equiv 3 \pmod{11}$ имеет два решения: $x \equiv 5 \pmod{11}$ и $x \equiv -5 \pmod{11}$. Но заметим, что $-5 \equiv 6 \pmod{11}$, так что фактически эти два решения 5 и 6. Также обратите внимание, что эти два решения неконгруэнтны (несравнимы).

Пример 13.2

Уравнение $x^2 \equiv 2 \pmod{11}$ не имеет решения. Не может быть найдено ни одного целого числа x, такого, что квадрат равен $2 \mod 11$.

Квадратичные вычеты и невычет

В уравнении $x^2 \equiv a \pmod{p}$. a называется квадратичным вычетом (QR), если уравнение имеет два решения; a называется квадратичным невычетом (QNR), если уравнение не имеет решений. Может быть доказано, что в Z_{p^*} с p - 1 элементами (p - 1)/2 элементов — квадратичные вычеты и (p - 1)/2 являются квадратичными невычетами.

Пример 13.3

Есть 10 элементов в Z_{11^*}. Пять из них – квадратичные вычеты, и пять — невычеты. Другими словами, Z_{11^*} может быть разделен на два отдельных множества, QR и QNR, как это показано на рис. 13.1.

Критерий Эйлера

Как мы можем проверить, является ли целое число QR по модулю p? Критерий Эйлера дает признаки:

a. Если $a^{(p-1)/2} \equiv 1 (\mathrm{mod} p)$ — квадратичный вычет по модулю p.

b. Если $a^{(p-1)/2} \equiv -1 (\mathrm{mod} p)$ — квадратичный невычет по модулю p.

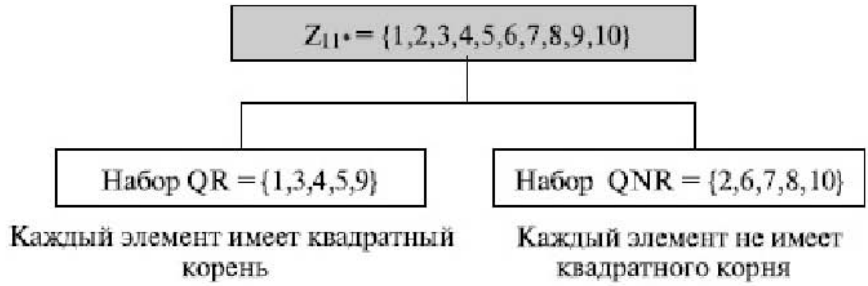


Рис. 13.1. Разделение Z11* на QR и QNR

Пример 13.4

Для того чтобы узнать, является ли 14 или 16 QR в Z_{11^*}, сделаем следующие вычисления:

$14^{(23-1)/2} \bmod 23$ $14^{11} \bmod 23$ -> $22 \bmod 23$ -> $-1 \bmod 23$ невычет
$16^{(23-1)/2} \bmod 23$ $16^{11} \bmod 23$ -> $1 \bmod 23$ вычет

Решение квадратичного сравнения с модулем в виде простого числа

Хотя критерий Эйлера позволяет нам определить, является ли целое число a QR или QNR в Z_{p^*}, он не может найти решение $x^2 \equiv (\bmod\ p)$. Чтобы найти решение этого квадратного уравнения, мы заметим, что простое число может быть представлено либо как p = 4k + 1, либо как p = 4 к + 3, в котором k является положительным целым числом. Решение квадратного уравнения — очень сложное в первом случае и более простое во втором. Мы обсудим только второй случай, который мы будем использовать в лекциях 14-15, когда будем рассматривать криптографическую систему Рабина.

Специальный случай: p = 4 K + 3, если p находится в форме 4 K + 3 (то есть p = 3 mod 4) и a есть QR в Z_{p^*}, то

$$x \equiv a^{(p+1)/4} (mod\ p) \text{и} x \equiv -a^{(p+1)/4} (mod\ p)$$

Пример 13.5

Решите следующие квадратные уравнения:

a. $x^2 \equiv 3\ (\bmod\ 23)$

b. $x^2 \equiv 2\ (\bmod\ 11)$

c. $x^2 \equiv 7\ (\bmod\ 19)$

Решения

a. В первом уравнении 3 – QR в Z_{23}, решение - $x \equiv \pm 16 \pmod{23}$. Другими словами, $\sqrt{3} \equiv \pm 16 \pmod{23}$.

b.Во втором уравнении 2 – QNR в Z_{11}. Нет решения для $\sqrt{2}$ в Z_{11}.

c. В третьем уравнении 7 – QR в Z_{19}, решение - $x \equiv \pm 11 \pmod{19}$. Другими словами $\sqrt{7} \equiv \pm 11 \pmod{19}$.

Квадратичное сравнение по составному модулю

Квадратичное сравнение по составному модулю может быть приведено к решению системы сравнений по модулю в виде простого числа. Другими словами, мы можем анализировать $x^2 \equiv a \pmod{n}$, если имеем разложение n на множители. Теперь мы можем решить каждое анализируемое уравнение (если оно *разрешимо*) и найти k пар ответов для x, как показано на рис. 13.2.

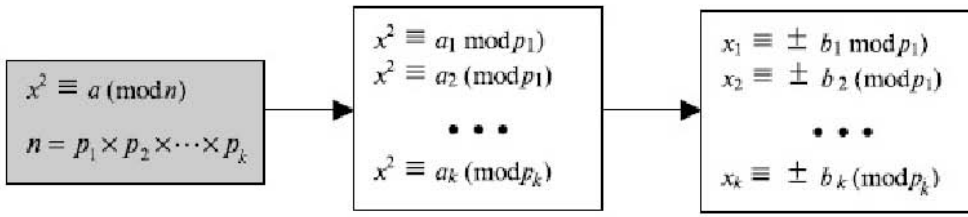


Рис. 13.2. Декомпозиция сравнения по составному модулю

Из k пар ответов мы можем составить 2 системы уравнений, которые могут быть решены с использованием китайской теоремы об остатках, чтобы найти 2 значения для x. В криптографии обычно n выбирают так, чтобы $n = p \times q$, — это означает k = 2, и мы имеем в целом только четыре ответа.

Пример 13.6

Предположим, что $x^2 \equiv 36 \pmod{77}$. Мы знаем, что $77 = 7 \times 11$.

Мы можем написать

$$x^2 = 36(mod\ 7) \equiv 1(mod\ 7) и x^2 \equiv 36(mod\ 11)$$

Обратите внимание, что мы выбрали `3` и `7`, чтобы иметь форму `4k + 3` — так, чтобы мы могли решить уравнения, основываясь на предыдущих рассуждениях. Из этих уравнений мы имеем квадратичные вычеты в собственном множестве. Ответы $x \equiv +1 \pmod 7$, $x \equiv -1 \pmod 7$, $x \equiv +5 \pmod{11}$ и $x \equiv -5 \pmod{11}$. Теперь мы можем из них составить четыре системы уравнений:

$$\begin{aligned}
&\texttt{Система 1: } x \equiv +1(mod\ 7) \\
&x \equiv +5(mod\ 11) \texttt{ Система 2: } x \equiv +1(mod\ 7) \\
&x \equiv -5(mod\ 11) \texttt{ Система 3: } x \equiv -1(mod\ 7) \\
&x \equiv +5(mod\ 11) \texttt{ Система 4: } x \equiv -1(mod\ 7) \\
&x \equiv -5(mod\ 11)
\end{aligned}$$

Ответы : $x = \pm 6$ и ± 27.

Сложность

Как сложно решить квадратичное сравнение по составному модулю? Главная задача — это разложение модуля на множители. Другими словами, сложность решения квадратичного сравнения по составному модулю — такая же, как и разложения на множители составного целого числа. Как мы видели раньше, если `n` очень большое, то разложение на множители неосуществимо.

Сложность решения квадратичного сравнения по составному модулю имеет ту же сложность, что и разложение модуля на множители.

13.2. Возведение в степень и логарифмы

Возведение в степень и логарифм инверсны друг другу. Следующие разделы показывают отношения между ними, в которых `a` называется основой возведения в степень или логарифма.

Возведение в степень: $y = a^x$ -> логарифм: $x = \log_a y$

Возведение в степень

В криптографии общая модульная операция — возведение в степень. Мы часто должны вычислять

$$y = a^x \bmod n$$

Криптографическая система RSA, которая будет обсуждаться влекциях 14-15, использует возведение в степень для шифрования и для *дешифрования* очень больших чисел. К сожалению, большинство компьютерных языков не имеет операторов, которые могут эффективно вычислять степень, особенно для очень больших чисел. Чтобы сделать эту операцию более эффективной при вычислении, мы нуждаемся в эффективных алгоритмах.

Быстрое возведение в степень

Быстрое возведение в степень возможно при использовании специальных методов возведения в квадрат и умножения. В традиционных алгоритмах, чтобы возводить в степень, применяется только умножение, но быстрый алгоритм возведения в степень использует и возведение в квадрат, и умножение. Главная идея этого метода — выполнение возведения в степень с помощью обработки двоичного числа с n_b битами (x_0 до x_{n_b-1}). Например, $x = 22 = (10110)_2$. Вообще, число x может быть записано как

$$x = x_{n_b-1} \times 2^{k-1} + x_{n_b-2} \times 2^{k-2} + \cdots + x_2 \times 2^2 + x_1 \times 2^1 + x_0 \times 2^0$$

Теперь мы можем написать $y = a^x$, как это показано на рис. 13.3.

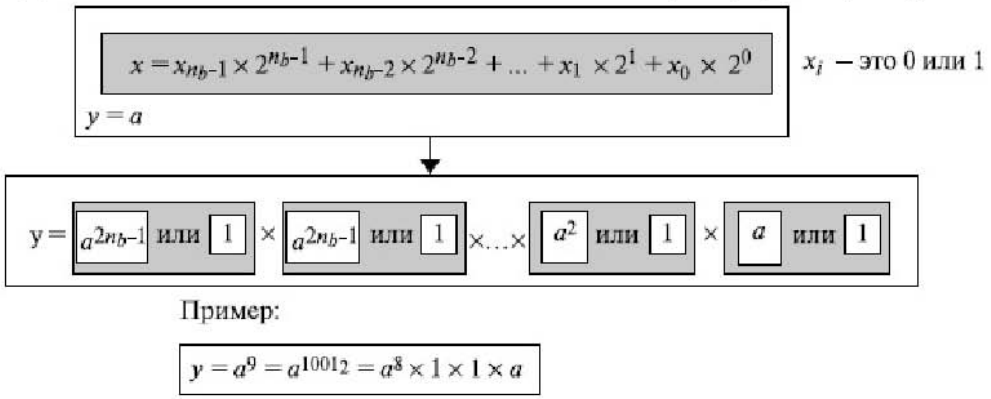


Рис. 13.3. Идея метода "возведения в квадрат и умножения"

Обратите внимание, что `y` — произведение элементов числа `n`. Каждый элемент — либо `1` (если соответствующий бит — `0`), либо a^{2i} (если соответствующий бит — `1`). Другими словами, элемент a^{2i} участвует в умножении, если бит — `1`, или не участвует, если бит — `0` (умножение на `1` не меняет числа). Рисунок 13.3 дает общую идею, как написать алгоритм. Мы можем непрерывно взводить в квадрат $a, a^2, a^4 \ldots a^{2^{n_b-1}}$ Если соответствующий бит — `0`, элемент не участвует в процессе умножения; если бит — `1`, то участвует. Алгоритм 13.1 отражает эти два свойства.

```
Возведение_в_ квадрат_и_умножение (a,x,n)
{
 y <- 1
 for (i <- 0 to n_b - 1)        //n_b — число бит в x
 {
  if (x_i = 1)   y <- a x y mod n      //умножение, только если бит 1
  a <- a^2 mod n
 }              //в последней итерации возведение в степень не нужно
 return y
}
```

Пример 13.1. Алгоритм 13.1

Алгоритм 13.1 использует n_b итерации. В каждой итерации он проверяет значение соответствующего бита. Если значение бита равно

1, он умножает текущее значение на предыдущее значение результата. Затем полученный результат является базой для следующей итерации. Обратите внимание, что возведение в квадрат в последнем шаге не нужно (результат не используется).

Пример 13.7

Рисунок 13.4 показывает процесс для подсчета $y = a^x$ с использованием алгоритма 13.1 (для простоты изображения модули не показаны). В этом случае $x = 22 = (10110)_2$. Это число имеет 5 бит.

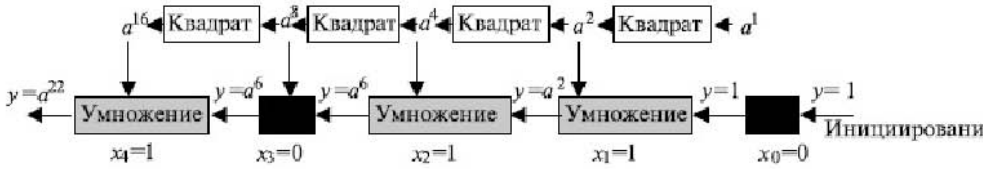


Рис. 13.4. Демонстрация вычисления a22 с использованием метода "возведения в квадрат и умножения"

Возведение в квадрат делается на каждом шаге за исключением последнего. Умножение делается, если соответствующий бит равен 1. На рисунке 13.4 показано, как постепенно формируется значение y до значения $y = a^{22}$. Затемненный прямоугольник означает, что умножение не делается и предыдущее значение переносится на следующий шаг. Таблица 13.1 показывает, как вычисляется значение для $y = 17^{22} \bmod 21$. Результат — $y = 4$.

Таблица 13.1.

i	x_i	Умножение (Инициализация y = 1)	Возведение в степень (Инициализация a = 17)
0	0		$a = 17^2 \bmod 21 = 16$
1	1	$y = 1 \times 16 \bmod 21 = 16$ →	$a = 16^2 \bmod 21 = 4$
2	1	$y = 16 \times 4 \bmod 21 = 1$ →	$a = 4^2 \bmod 21 = 16$
3	0		$a = 16^2 \bmod 21 = 4$
4	1	$y = 1 \times 4 \bmod 21 = 4$ →	

Сложность. Алгоритм 13.1 использует максимально $2n_b$ арифметических операций, в которых n_b является длиной модуля в битах ($n_b = \log_2 n$). Сложность в *битовых операциях* алгоритма — $O(n_b)$ или полиномиальная сложность.

Сложность разрядной операции быстрого показательного алгоритма — полиномиальная.

Альтернативный алгоритм. Заметим, что алгоритм 13.1 проверяет значение битов в x справа налево (от самого младшего до самого старшего). Может быть написан другой алгоритм, чтобы использовать обратный порядок. Мы выбрали вышеупомянутый алгоритм, потому что операция возведения в квадрат полностью независима от операции умножения; они могут быть сделаны параллельно, чтобы увеличить скорость обработки. Альтернативные алгоритмы оставляем как упражнение.

Логарифм

Мы также должны обсудить модульный логарифм, который используется в криптографии. Если мы применяем возведение в степень для того, чтобы зашифровать или расшифровывать сообщение, противник может использовать логарифм для раскрытия этого сообщения Мы должны знать, трудно ли получить операцию, противоположную возведению в степень.

Полный перебор

Первое решение, которое могло бы прийти на ум, для решения $x = \log_a y \pmod n$: мы можем написать алгоритм, который непрерывно вычисляет $y = a^x \bmod n$, пока не находит заданное значение y. Алгоритм 13.2 показывает этот подход.

```
Modular_Logarithm (a, y, n}
{
  for (x = 1 to n - 1) // k число бит в x
  {
    if (y ≡ a^x mod n) return x
  }
  return failure
}
```

Пример 13.2. Алгоритм 13.2. Алгоритм полного перебора для модульного логарифма

Алгоритм 13.2 явно очень неэффективен. Сложность разрядной операции — $O(2^n)$ или показательна.

Дискретный логарифм

Второй подход состоит в том, чтобы использовать понятие дискретного логарифма. Рассмотрение этого понятия требует изучения некоторых свойств мультипликативных групп.

Конечная мультипликативная группа. В криптографии мы часто используем мультипликативную конечную группу: $G = Z_{n*}, \times$, в которой применяемая операция является умножением. Множество Z_{n*} содержит целые числа от 1 до n-1, которые являются взаимно простыми с n, нейтральный элемент — e = 1. Обратите внимание, что когда модуль группы — простое число, мы имеем $G = Z_{p*}, \times$. Эта группа — специальный случай группы первого типа, так что мы концентрируемся на этой группе.

Порядок группы. В лекциях 5-6 мы обсуждали порядок конечной группы |G| для того, чтобы определить число элементов в группе G. Было доказано, что в группе $G = Z_{n*}, \times$ порядок группы (число элементов) — *число Эйлера* $\varphi(n)$. Мы показали, как вычислить $\varphi(n)$, когда n может быть разложено на множители в виде простых чисел.

Пример 13.8

Найти порядок группы $G = Z_{21*}, \times$, $|G| = \varphi(21) = \varphi(3) \times \varphi(7) = 2 \times 6 = 12$. В этой группе 12 элементов: 1, 2, 4, 5, 8, 10, 11, 13, 16, 17, 19 и 20. Все — взаимно простые с 21.

Порядок элемента. В лекциях 5-6 мы также обсуждали порядок элемента — ord (a). В $G = Z_{n*}, \times$, мы продолжаем определение. Порядок элемента a есть наименьшее целое число, такое, что $a^i \equiv e(\mathrm{mod} n)$. В этом случае нейтральный элемент e равен 1.

Пример 13.9

Найдите порядок всех элементов в $G = Z_{10*}, \times$.

Решение

Эта группа имеет только $\varphi(10) = 4$ элемента: 1, 3, 7, 9. Мы можем найти порядок каждого элемента методом "проб и ошибок". Однако, согласно результатам лекций 5-6, порядок элемента является делителем порядка группы (теорема Лагранжа). Целые числа, которые делят 4, − 1, 2 и 4. Это означает, что в каждом случае мы должны проверить только эти числа и найти порядок элемента,

a. $1^1 = 1 \bmod (10) \rightarrow ord\,(10) = 1$.

b.

$3^1 = 3 \bmod (10); 3^2 \equiv 9 \bmod (10); 3^4 \equiv 1 \bmod (10) \rightarrow ord\,(3) = 4$

.

c.

$7^1 = 7 \bmod (10); 7^2 \equiv 9 \bmod (10); 7^4 \equiv 1 \bmod (10) \rightarrow ord\,(7) = 4$

d. $9^1 = 9 \bmod (10); 9^2 \equiv 1 \bmod (10) \rightarrow ord\,(9) = 2$.

Теорема Эйлера. Другая теорема, относящаяся к этому вопросу, — это

теорема Эйлера, которая говорит, что если a является членом $G = Z_{n^*}, \times$, то $a^{\varphi(n)} = 1 \bmod n$;

Эта теорема очень полезна, потому что показывает, что равенство $A^i = 1 \bmod n$ сохраняется, если $i = \varphi(n)$. Заметим: это не отрицает, что равенство может выполняться и если $i < \varphi(n)$. Другими словами, это отношение показывает, что равенство выполняется по меньшей мере однажды.

Пример 13.10

Таблица 13.2 показывает результат $a^i = x \pmod 8$ для группы $G = Z_{8^*}, \times$. Обратите внимание, что $\varphi(8) = 4$. Это элементы — 1, 3, 5 и 7.

Таблица 13.2. Нахождение порядка элементов в примере 13.10

	i=1	i=2	i=3	i=4	i=5	i=6	i=7
a=1	x:1	x:1	x:1	x:1	x:1	x:1	x:1
a=3	x:3	x:1	x:3	x:1	x:3	x:1	x:3
a=5	x:5	x:1	x:5	x:1	x:5	x:1	x:5
a=7	x:7	x:1	x:7	x:1	x:7	x:1	x:7

Таблица 13.2 показывает некоторые моменты вычислений. Первый: затемненная область показывает результат применения *теоремы Эйлера*. Когда $i = \varphi(8) = 4$, результат равен x = 1 для каждого a. Второй момент: когда таблица показывает, что значение 1 может быть получено для многих значений i — в первую очередь, значение i, равное порядку элемента (обведено в таблице жирной линией). Порядок элементов: ord (1) = 1, ord (3) = 2, ord (5) = 2 и ord (7) = 2.

Первообразные корни. Очень интересное понятие в мультипликативной группе — группы первообразного корня, которые используются в *криптографической системе* El Gamal (эль Гамаля) в лекциях 14-15. В

группе $G = Z_{n^*}, \times$, когда порядок элемента равен $\varphi(n)$, этот элемент называется первообразным корнем группы.

Пример 13.11

Таблица 13.2 показывает, что там нет первообразных корней $G = Z_{8^*}, \times$, потому что ни один элемент не имеет порядок, равный $\varphi(8) = 4$. Порядок всех элементов — меньше, чем 4.

Пример 13.12

Таблица 13.3 показывает результат $a^i \equiv x \pmod 7$ для группы $G = Z_{7^*}, \times$. В этой группе $\varphi(7) = 6$.

Таблица 13.3. Пример 13.12

	i=1	i=2	i=3	i=4	i=5	i=6
a=1	x:1	x:1	x:1	x:1	x:1	x:1
a=2	x:2	x:4	x:1	x:2	x:4	x:1
a=3	x:3	x:2	x:6	x:4	x:5	x:1
a=4	x:4	x:2	x:1	x:4	x:2	x:1
a=5	x:5	x:4	x:6	x:2	x:3	x:1
a=6	x:6	x:1	x:6	x:1	x:6	x:1

Порядок элементов – $ord(1) = 1$, $ord(2) = 3$, $ord(3) = \underline{6}$, $ord(4) = 3$, $ord(5) = \underline{6}$ и $ord(6) = 1$. Таблица 13.3 показывает, что только два элемента, 3 и 5, имеют порядок в $i = \varphi(n) = 6$. Поэтому эта группа имеет только два примитивных корня: 3 и 5.

Было доказано, что группа $G = Z_{n^*}, \times$ имеет примитивный корень, только если $n = 2, 4, p^t$, или $2p^t$, в которой p является нечетным простым числом (не 2) и t — целое число.

Группа G = < Z_{n^*} , x > имеет примитивные корни, только если n равно

2, 4, p^t или $2p^t$.

Пример 13.13

Для какого значения n группа $G = Z_{n^*}, \times$ имеет примитивные корни: 17, 20, 38 и 50?

Решение

a. $G = Z_{17^*}, \times$ имеет примитивные корни, потому что 17 — простое число (p^t, где t равно 1).

b. $G = Z_{20^*}, \times$ не имеет никаких примитивных корней.

c. $G = Z_{38^*}, \times$ имеет *первообразные* корни, потому что $38 = 2 \times 19$ и 19 — простое число.

d. $G = Z_{50^*}, \times$ имеет *первообразные* корни, потому что $50 = 2 \times 5^2$, а 5 — простое число.

Если группа имеет примитивный корень, то обычно она имеет несколько таких корней. Число примитивных корней может быть вычислено как — $\varphi(\varphi(n))$. Например, число примитивных корней $G = Z_{17^*}, \times$ — это — $\varphi(\varphi(n)) = \varphi(16) = 8$. Обращаем внимание, что нужно сначала проверить, имеет ли группа какой-либо примитивный корень, прежде чем находить число корней.

Если группа G = < Z_{n^*} , x > имеет хотя бы один примитивный корень, то число примитивных корней — ϕ (ϕ (n))

Рассмотрим три вопроса:

1. Если дан элемент a и группа $G = Z_{n^*}, \times$, как можно определить, является ли a примитивным корнем G? Это не такая легкая задача.

a. Мы должны найти $\varphi(n)$, — эта задача по сложности подобна задаче разложения на множители числа n.

б. Мы должны найти $ord(a) = \varphi(n)$.

2. Если дана группа $G = Z_{n^*}, \times$, как найти все примитивные корни φ? Эта задача более трудная, чем первая задача, потому что мы должны повторить вычисления по п.1.б для всей группы.

3. Если дана группа $G = Z_{n^*}, \times$, то как выбирать примитивный корень G? В криптографии мы должны найти, по крайней мере, один примитивный корень в группе. Однако в этом случае значение n выбирается пользователем, и пользователь знает $\varphi(n)$. Пользователь пробует последовательно несколько элементов, пока не находит первый из них.

Циклическая группа. Циклические группы уже обсуждались в лекциях 5-6. Обратите внимание на то, что, если группа $G = Z_{n^*}, \times$ имеет примитивные корни, то они циклически повторяются. Каждый примитивный корень — генератор и может использоваться для создания целого набора. Другими словами, если g — примитивный корень в группе, мы можем генерировать набор Z_n^* как

$$Z_n^* = \{g^1, g^2, g^3, \ldots, g^{\varphi(n)}\}$$

Пример 13.14

Группа $G = Z_{10^*}, \times$ имеет два примитивных корня, потому что $\varphi(10) = 4$ и $\varphi(\varphi(10)) = 2$. Можно найти примитивные корни - это 3 и 7. Ниже показано, как можно создать целый набор Z_{10^*}, использующий каждый примитивный корень.

g = 3 -> $g^1 \bmod 10 = 3$ $g^2 \bmod 10 = 9$ $g^3 \bmod 10 = 7$ $g^4 \bmod 10 = 1$
g = 7 -> $g^1 \bmod 10 = 7$ $g^2 \bmod 10 = 9$ $g^3 \bmod 10 = 3$ $g^4 \bmod 10 = 1$

Обратите внимание, что группа $G = Z_{p^*}, \times$ всегда циклическая, потому что p — простое.

Группа G = < Z_n^*, x > является циклической группой, если она имеет пр

Группа G = < Z_p^*, x > всегда является циклической.

Идея дискретного логарифма. Группа $G = Z_p^*, \times$ имеет несколько интересных свойств.

1. Её элементы включают все целые числа от 1 до p – 1.
2. Она всегда имеет примитивные корни.
3. Она всегда является циклической. Элементы могут быть созданы с использованием g^x, где x — целое число от 1 до $\varphi(n) = p - 1$.
4. Примитивные корни можно представлять себе как основание логарифма. Если группа имеет k примитивных корней, то вычисления могут быть сделаны для k различных оснований. Данный $x = \log_g y$ для любого элемента y в данном множестве, но есть другой элемент x, который является логарифмом y по основанию g. Этот тип логарифма называют дискретным логарифмом. *Дискретный логарифм* в литературе определяется несколькими различными символами, но мы будем использовать обозначение L_g, чтобы показать, что основание — g (сравнение по модулю).

Решение модульного логарифма с использованием дискретных логарифмов

Теперь рассмотрим, как решаются задачи типа $y = a^x \pmod n$, т. е. дано y, а мы должны найти x.

Табулирование дискретных логарифмов. Один из способов решения вышеупомянутой проблемы — использовать таблицу для каждого Z_{p*} и различных оснований. Этот тип таблицы может быть предварительно рассчитан и сохранен. Например, таблица 13.4 показывает значения *дискретного логарифма* для Z_{7*}. Мы знаем, что мы имеем два примитивных корня или основания в данном множестве.

Таблица 13.4. *Дискретный логарифм* для G =

<Zp*,x>

y	1	2	3	4	5	6
$x = L_3 y$	6	2	1	4	5	3
$x = L_5 y$	6	4	5	2	1	3

Составив таблицы для других *дискретных логарифмов* для всех групп и всех возможных оснований, мы можем решить любую дискретную логарифмическую проблему. Этот подход подобен изучаемым в прошлом традиционным логарифмам. До появления калькуляторов и компьютеров таблицы использовались, чтобы вычислять логарифмы по основанию 10.

Пример 13.15

Найдите x в каждом из следующих случаев:

a. $4 \equiv 3^x \pmod 7$

b. $6 \equiv 5^x \pmod 7$

Решение

Мы можем легко использовать таблицу 13.4 *дискретного логарифма*.

a. $4 \equiv 3^x \pmod 7 \rightarrow L_3 4 \pmod 7 = 4 \mod 7$

b. $6 \equiv 5^x \pmod 7 \rightarrow L_5 6 \pmod 7 = 3 \mod 7$

Использование свойств дискретных логарифмов. Чтобы показать, что *дискретные логарифмы* ведут себя точно так же, как традиционные логарифмы, в таблице 13.5 приводится несколько свойств обоих типов логарифмов. Обратите внимание, что основание модуля — $\varphi(n)$ вместо n.

Таблица 13.5. Сравнение традиционных и *дискретных логарифмов*

Традиционные логарифмы	*Дискретные логарифмы*

$\log_a 1 = 0$	$L_g \equiv 0 (\mathrm{mod}\ \phi (n))$
$\log_a (x \times y) = \log_a x + \log_a y$	$L_g (x \times y) \equiv L_g x + L_g y$
$\log_a x^k = k \times \log_a x$	$L_g x \equiv k \times L_g x (\mathrm{mod}\ \phi (n))$

Использование алгоритмов, основанных на дискретных логарифмах. Таблицы и свойства *дискретных логарифмов* не могут быть использованы для решения уравнения $y = a^x \pmod n$, когда n является очень большим. Для решения этой проблемы были разработаны несколько алгоритмов, в которых используется основная идея *дискретных логарифмов*. Хотя все эти алгоритмы более эффективны, чем алгоритмы полного перебора, которые мы упоминали в начале этого раздела, но ни один из них не имеет полиномиальную сложность. Большинство этих алгоритмов имеет такой же уровень сложности, как проблема разложения на множители.

Проблема дискретного логарифма имеет такую же сложность, как проблема разложения на множители.

13.3. Рекомендованная литература

Нижеследующие книги и сайты дают более детальную информацию о предметах, рассмотренных в этой лекции. Пункты, приведенные в квадратных скобках, содержатся в списке в конце книги.

Книги

[Ros06] [Cou99], [BW00] и [Bla03] — для тем, которые обсуждаются в этой лекции.

Сайты

Нижеследующие сайты дают больше информации о темах, обсужденных в этой лекции.

- ссылка: http://en.wikipedia.org/wiki/Prime_number - http://en.wikipedia.org/wiki/Prime_number
- ссылка: http://primes.utm.edu/mersenne/ - http://primes.utm.edu/mersenne/
- ссылка: http://en.wikipedia.org/wiki/Primality__test - http://en.wikipedia.org/wiki/Primality__test
- ссылка: www..cl.cam.ac.uk/~jehl004/research/talks/miller-talk.pdf - http://www..cl.cam.ac.uk/~jehl004/research/talks/miller-talk.pdf
- ссылка: http://mathworld.wolfram.com/TotientFunction.html - http://mathworld.wolfram.com/TotientFunction.html
- ссылка: http: // en.wikipedia.org/wiki/Proofs_of_Fermat's_little_theorem - http://en.wikipedia.org/wiki/Proofs_of_Fermat's_little_theorem
- ссылка: faculty.cs.tamu.edu/klappi/629/analytic.pdf - faculty.cs.tamu.edu/klappi/629/analytic.pdf

13.4. Итоги

- Положительные целые числа могут быть разделены на три группы: число `1`, простые числа и *составные объекты*. Положительное целое число — простое число, такое и только такое, если оно точно делится без остатка на два различных целых числа, а именно на `1` и непосредственно само на себя. Составной объект — положительное целое число по крайней мере с двумя делителями.
- Эйлеровская `phi` -функция $\varphi(n)$, которую иногда называют функцией-тотиентом Эйлера, играет очень важную роль в криптографии. Функция указывает число целых чисел, которые меньше чем `n` и являются взаимно простыми с `n`.
- В таблице 13.6 показаны малая *теорема Ферма* и *теорема Эйлера*, которые рассмотрены в этой лекции.

Таблица 13.6. Малая *теорема Ферма* и *теорема Эйлера*

Ферма	Первая версия : Если НОД(a,p) = 1, то $a^{p-1} \equiv 1 (\text{mod } p)$
	Вторая версия: $a^{p} \equiv a (\text{mod } p)$
Эйлер	Первая версия: Если НОД(a,n) = 1, то $a^{\phi(n)} \equiv 1 (\text{mod } p)$
	Вторая версия: Если n= p x q и a < n, то $a>kx^{\phi(n)+1} \equiv a (\text{mod } n)$

- Чтобы получить большое простое число, мы выбираем большое

случайное число и проверяем его — убеждаемся, что оно простое. Алгоритмы, которые решают эту проблему, могут быть разделены на две обширные категории: детерминированные алгоритмы и вероятностные алгоритмы. Некоторые вероятностные алгоритмы для испытания простоты чисел – это испытание Ферма, испытание квадратного корня и испытание Миллера-Рабина. Некоторые детерминированные алгоритмы — испытание на *делимости* и AKS-алгоритм.

- Согласно основной теореме арифметики, любое положительное целое число, большее, чем `1`, может быть разложено на множители в виде простых чисел. Мы рассмотрели несколько методов разложения на множители, включая проверку делением — Ферма, метод Полларда `p - 1`, метод `PO` (ρ) Полларда, квадратичное решето и решето поля чисел.
- Китайская теорема об остатках (Chinese Reminder Theorem — *CRT*) используется, чтобы решить систему уравнений для вычетов с одной переменной, но с различными взаимно простыми модулями.
- Мы рассмотрели решение квадратичного сравнения по модулю в виде простого числа и квадратичного сравнения по составному модулю. Однако, если модуль является большим, решение квадратичного сравнения по сложности совпадает с разложением модуля на множители.
- В криптографии применяется операция по модулю — возведение в степень. Для быстрого возведения в степень можно использовать метод "возведения в квадрат и умножения". Криптография также включает модульные логарифмы. Если возведение в степень применяется, чтобы зашифровать или расшифровывать информацию, то противник может использовать логарифмы для организации "атаки". Сложность операции, обратной возведению в степень, велика. Хотя возведение в степень может быть сделано с помощью быстрого алгоритма, применение модульного логарифма для больших значений модуля имеет ту же сложность, что и проблема разложения на множители.

13.5. Набор для практики

Обзорные вопросы

1. Объясните разницу между простым числом и составным целым числом.
2. Определите взаимно простые числа и их свойства.
3. Определите следующие функции и их приложения:
   - $\pi(n)$ функция
   - Функция (тотиент) Эйлера
4. Объясните "*решето Эратосфена*" и его приложения.
5. Определите малую теорему Ферма и объяснить ее приложения.
6. Определите теорему Эйлера и объясните ее приложения.
7. Что такое простые *числа Мерсенны*? Что такое простые *числа Ферма*?
8. Объясните разницу между детерминированными и вероятностными алгоритмами для определения простых чисел.
9. Перечислите некоторые алгоритмы для разложения на множители простых чисел.
10. Определите Китайскую теорему об остатках и ее приложения.
11. Определите квадратичное сравнение и важность вычетов (QRs) и невычетов (QNRs) в решении квадратных уравнений.
12. Определите *дискретные логарифмы* и объяснить их важность в решении логарифмических уравнений.

Упражнения

1. Используя аппроксимацию, найдите:
   - число простых чисел между `100 000` и `200 000`.
   - число составных целых чисел между `100 000` и `200 000`
   - отношение простых чисел к составным в вышеупомянутом диапазоне и сравните это с тем же самым между `1 – 10`.
2. Найти наибольший простой сомножитель следующих составных целых чисел: `100, 1000, 10 000, 100 000` и `1 000 000`. Также найти наибольший простой сомножитель `101, 1001, 10 001, 100 001` и `1 000 01`.
3. Покажите, что каждое простое число может быть представлено в форме либо `4k + 1`, либо `4k + 3`, где `k` — положительное

целое число.

4. Найдите некоторые простые числа в форме `5k + 1`, `5k + 2`, `5 k + 3` и `5k + 4`, где `k`: является положительным целым числом.
5. Найдите значение $\varphi(29)$, $\varphi(32)$, $\varphi(80)$, $\varphi(100)$, $\varphi(101)$
6. Покажите, что $2^{24} - 1$ и $2^{16} - 1$ – составные числа. Подсказка: используйте выражение $(a^2 - b^2)$.
7. Есть предположение, что каждое целое число, большее, чем 2, может быть представлено как сумма двух простых чисел. Проверьте это предположение для 10, 24, 28 и 100.
8. Есть предположение, что есть много простых чисел в форме $n^2 + 1$. Найдите некоторые из них.
9. Найдите результаты после использования малой *теоремы Ферма*:
   - $5^{15} \bmod 13$
   - $15^{18} \bmod 17$
   - $456^{17} \bmod 17$
   - $145 \bmod 101$
10. Найдите, используя Малую теорему Ферма, результаты выражений, приведенных ниже:
    - $5^{-1} \bmod 13$
    - $15^{-1} \bmod 17$
    - $27^{-1} \bmod 41$
    - $70^{-1} \bmod 101$

    Обратите внимание, что все модули — простые числа.

11. Найдите, используя теорему Эйлера, результаты выражений, приведенных ниже:
    - $12^{-1} \bmod 77$
    - $16^{-1} \bmod 323$
    - $20^{-1} \bmod 403$
    - $44^{-1} \bmod 667$

    Обратите внимание, что $77 = 7 \times 11$, $323 = 17 \times 19$, $403 = 31 \times 13$ и $667 = 23 \times 29$.

12. Определите, являются ли следующие *числа Мерсенны* простыми числами: M_{23}, M_{29} и M_{31}. Подсказка: любой делитель *числа Мерсенны* имеет форму $2kp + 1$.
13. Приведите некоторые примеры, чтобы показать, что если $2^n - 1$ — простое число, то n — простое число. Этот факт может использоваться для проверки на простоту? Объясните, как.
14. Определите, сколько из следующих целых чисел пройдут испытание Ферма на простоту чисел: 100, 110, 130, 150, 200, 250, 271, 341, 561. Используйте основание 2.
15. Определите, сколько из следующих целых чисел пройдут испытание Миллера-Рабина на простоту чисел: 100, 109, 201, 271, 341, 349. Используйте основание 2.
16. Используйте рекомендованное испытание, чтобы определить, является ли любое из следующих целых чисел простым числом: 271, 3149, 9673.
17. Используйте $a = 2$, $x = 3$ и несколько простых чисел, чтобы показать, что если p — простое число, то выполняется следующее сравнение: $(x - a)^p = (x^p - a) \pmod p$.
18. Говорят, что n -ное простое число может быть приближенно вычислено как $p_n = n \ln n$. Проверьте это на нескольких простых числах.
19. Найдите значение x для следующих наборов сравнений, используя китайскую теорему об остатках.
    - $x \equiv 2 \bmod 7$, и $x \equiv 3 \bmod 9$
    - $x \equiv 4 \bmod 5$, и $x \equiv 10 \bmod 11$
    - $x \equiv 7 \bmod 13$, и $x \equiv 11 \bmod 12$
20. Найдите весь QRs и QNRs в Z_{13^*}, Z_{17^*} и Z_{23^*}.
21. Используя квадратичные вычеты, решите следующие сравнения:
    - $x^2 \equiv 4 \bmod 7$
    - $x^2 \equiv 5 \bmod 11$
    - $x^2 \equiv 7 \bmod 13$
    - $x^2 \equiv 12 \bmod 17$
22. Используя квадратичные вычеты, решите следующие сравнения:
    - $x^2 \equiv 4 \bmod 14$
    - $x^2 \equiv 5 \bmod 10$
    - $x^2 \equiv 7 \bmod 33$

- $x^2 \equiv 12 \bmod 34$

23. Найдите результаты приведенных ниже выражений, используя метод "возведения в квадрат и умножения".
    - $21^{24} \bmod 8$
    - $320^{23} \bmod 461$
    - $1736^{41} \bmod 2134$
    - $2001^{35} \bmod 2000$
24. Для группы $G = Z_{19^*}, \times$:
    - Найдите порядок группы
    - Найдите порядок каждого элемента в группе
    - Найдите число первообразных корней в группе
    - Найдите *первообразные* корни в группе
    - Покажите, что группа является циклической
    - Составьте таблицу *дискретных логарифмов*
25. Используя свойства *дискретных логарифмов*, покажите, как решить сравнения:
    - $x^5 \equiv 11 \bmod 17$
    - $2x^{11} \equiv 22 \bmod 19$
    - $5x^{12} + 6x \equiv 8 \bmod 23$
26. Пусть мы имеем компьютер, выполняющий операции со скоростью 1 миллион бит в секунду. Вы хотите затратить только 1 час на испытание простоты чисел. Какое наибольшее число вы можете проверить, используя следующие методы, проверяющие простоту чисел?
    - теория *делимости*
    - AKS-алгоритм
    - Ферма
    - извлечением квадратного корня
    - Миллера-Рабина
27. Пусть мы имеем компьютер, выполняющий операции со скоростью 1 миллион бит в секунду. Вы хотите потратить только 1 час на разложение составного целого числа. Какое наибольшее число вы можете разложить на множители, используя следующие методы разложения на множители?
    - проверка делением
    - Ферма
    - Полларда (P0)

- квадратичное решето
- решето поля чисел

28. Метод "возведения в квадрат и умножения" — быстрый алгоритм возведения в степень — позволяет нам останавливать программу, если значение основания становится равным `1`. Измените алгоритм 13.1, чтобы показать это.
29. Перепишите алгоритм 13.1, чтобы проверить биты в порядке от самого старшего к самому младшему.
30. Метод "возведения в квадрат и умножения" — быстрый алгоритм возведения в степень — может также быть спроектирован для проверки, является ли число четным или нечетным, вместо того чтобы проверять его разряды. Перепишите алгоритм 13.1, чтобы показать это.
31. Напишите алгоритм в псевдокоде для испытания простоты чисел по методу Ферма.
32. Напишите алгоритм в псевдокоде для испытания простоты чисел методом извлечения квадратного корня.
33. Напишите алгоритм в псевдокоде для китайской теоремы об остатках.
34. Напишите алгоритм в псевдокоде, чтобы найти вычет (QR) и невычет (QNR) для любого Z_{p^*}.
35. Напишите алгоритм в псевдокоде для нахождения *первообразного* корня для множества Z_{p^*}.
36. Напишите алгоритм в псевдокоде, чтобы найти все *первообразные* корни для множества Z_{p^*}.
37. Напишите алгоритм, чтобы найти и хранить *дискретные логарифмы* для множества Z_{p^*}.

Криптографическая система RSA

В этой лекции рассматривается асимметрично-ключевая криптографическая система: RSA (RIVERST-SHAMIR-ADLEMAN).

14.1. Введение

В лекциях 2-11 мы показали принципы криптографии с симметричными ключами. В этой лекции мы начинаем обсуждение асимметрично-ключевой криптографии. Симметричная и асимметрично-ключевая криптографии будут существовать параллельно и продолжать обслуживать общество. Мы верим, что они дополняют друг друга; преимущества одной компенсируют недостатки другой.

Концептуальные отличия между этими двумя системами базируются на том, как эти системы сохраняют секретность. В криптографии с *симметричными ключами* задача секретности должна быть разделена между двумя людьми. В асимметрично-ключевой криптографии секретность — персональная задача (неразделенная); человек создает и сохраняет свою собственную тайну.

В сообществе `n` людей при криптографии с *симметричными ключами* для сохранения секретности требуется `n (n – 1)/2` общедоступных ключей. В асимметрично-ключевой криптографии необходимы только `n` персональных ключей. Сообщество с количеством участников `1` миллион при криптографии с *симметричными ключами* требовало бы пятисот миллионов общедоступных ключей; асимметрично-ключевая криптография требовала бы `1` миллион персональных ключей.

Криптография с симметричными ключами базируется на совместном использовании ключей; асимметрично-ключевая криптография базируется на персональном ключе.

Есть некоторые другие аспекты безопасности помимо шифрования, которые присущи асимметрично-ключевой криптографии. Они включают установление подлинности и цифровые подписи. Всякий раз, когда приложение базируется на персональной тайне, мы должны использовать асимметрично-ключевую криптографию.

Обратим внимание на то, что криптография с *симметричными ключами* базируется на подстановке и перестановке символов (символов или бит), а асимметрично-ключевая криптография — на применении математических функций к числам. В криптографии с *симметричными ключами* исходный текст и зашифрованный текст представляют как комбинацию символов. Шифрование и дешифрование здесь — это перестановка этих символов или замена одного символа другим. В асимметрично-ключевой криптографии исходный текст и зашифрованный текст - числа; их шифрование и дешифрование — это *математические функции*, которые применяются к числам, чтобы создать другие числа.

В криптографии с симметричными ключами, символы переставляются или заменяются другими; в асимметрично-ключевой криптографии числа преобразуются с помощью математических функций.

Ключи

Асимметричная ключевая криптография использует два отдельных ключа: один секретный (частный) и один открытый (общедоступный); шифрование и дешифрование представляют процесс запирания и отпирания замков ключами. В этом случае замок, запертый открытым ключом доступа, можно отпереть только с соответствующим секретным ключом. Рисунок 14.1 показывает, что если Алиса запирает замок открытым ключом доступа Боба, то только секретный ключ Боба может отпереть его.

Общая идея

Рисунок 14.2 показывает общую идею асимметрично-ключевой криптографии при использовании для шифрования. В будущих лекциях мы увидим другие приложения асимметрично-ключевой криптографии. Как показывают рисунки, в отличие от криптографии с *симметричными ключами* при асимметрично-ключевой криптографии ключи отличаются: существует секретный ключ и открытый ключи доступа. Хотя в некоторых книгах используют термин ключ засекречивания вместо термина секретный ключ, мы будем

пользоваться термином ключ засекречивания только для системы с *симметричными ключами* и термином секретный ключ и открытый ключ доступа — для асимметрично-ключевой криптографии. Мы даже применим различные символы, чтобы показать три различных типа ключей. Одна из причин разницы в терминах — характер ключа засекречивания, используемого в криптографии с симметричными ключами, отличается от характера секретного ключа, применяемого в асимметрично-ключевой криптографии.

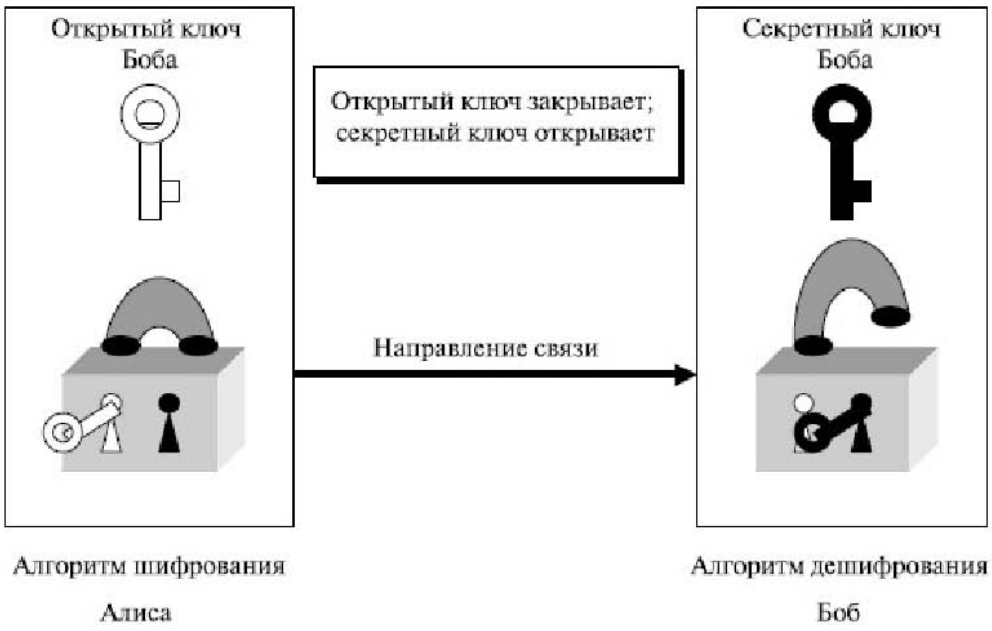


Рис. 14.1. Закрытие и открытие в асимметрично - ключевой криптосистеме

Первый ключ работает обычно со строкой символов (например, биты), второй — с числами или множеством чисел. Другими словами, мы хотим показать, что ключ засекречивания не является взаимозаменяемым с секретным ключом ; это два различных типа секретности. Рисунок 14.2 иллюстрирует несколько важных фактов.

Первый: подчеркивает асимметричный характер криптографической системы. Ответственность за обеспечение безопасности находится, главным образом, на плечах приемника (в данном случае это Боб). Боб должен создать два ключа: один секретный (частный) и один открытый

(общедоступный). Боб не несет ответственность за распределение открытого ключа доступа всему сообществу. Это может быть сделано через канал распределения открытого ключа доступа. Хотя этот канал не обязан обеспечивать секретность, он должен обеспечить установление подлинности и целостность информации о ключе. Ева не должна иметь возможности распространять свой открытый ключ сообществу, представляя его как открытый ключ доступа Боба. На данный момент мы принимаем, что такой канал существует.

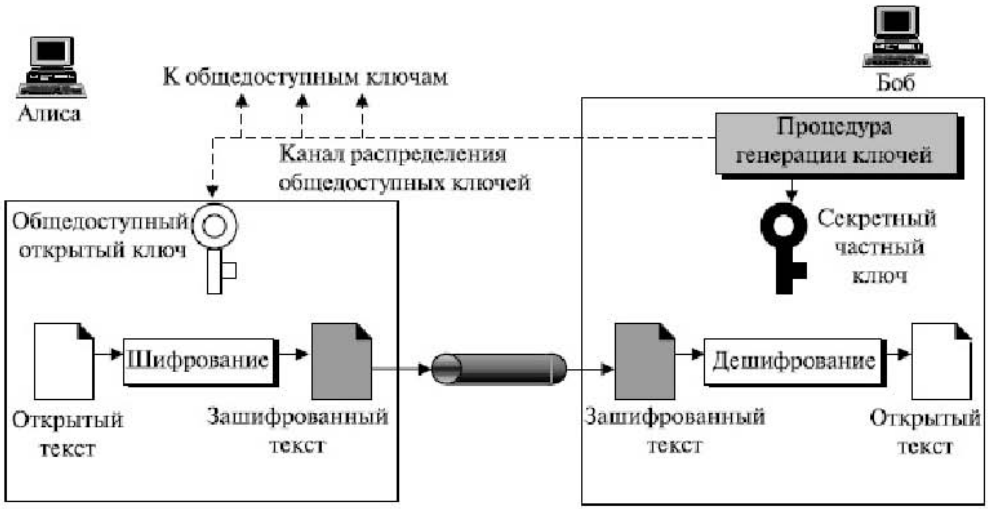


Рис. 14.2. Общая идея асимметрично-ключевой криптосистемы

Второй факт: асимметрично-ключевая криптография означает, что Боб и Алиса не могут использовать одно и то же множество ключей для двухсторонней связи. Каждый объект в сообществе создает свой собственный секретный и открытый ключи доступа. Рисунок 14.2 показывает, как Алиса может использовать открытый ключ доступа Боба, чтобы передать Бобу зашифрованные сообщения. Если Боб хочет ответить, Алиса устанавливает свои собственные секретный и открытый ключи доступа.

Третий: асимметрично-ключевая криптография означает, что Боб нуждается только в одном секретном ключе, чтобы получать всю корреспонденцию от любого участника сообщества. Алиса нуждается в ключах, чтобы связаться с n объектами в сообществе — один ключ доступа для каждого. Другими словами, Алиса нуждается в *кольце*

ключей доступа.

Исходный текст / зашифрованный текст

В отличие от криптографии с *симметричными ключами*, в асимметрично-ключевой криптографии исходный текст и зашифрованный текст обрабатываются как целые числа. Сообщение должно перед шифрованием кодироваться как целое число (или множество целых чисел). После дешифрования оно должно быть расшифровано как целое число (или множество целых чисел). Асимметрично-ключевая криптография обычно зашифровывает или расшифровывает маленькие части информации, определяемые *длиной ключа* шифра. Другими словами, асимметрично-ключевая криптография обычно имеет вспомогательные цели помимо шифровки сообщения. Однако эти вспомогательные цели сегодня играют в криптографии очень важную роль.

Шифрование/дешифрование

Шифрование и дешифрование в асимметрично-ключевой криптографии — *математические функции*, которые применяются к числам, представляющим исходный текст и зашифрованный текст. Зашифрованный текст можно представлять себе как `C = f (K public, P)`. Исходный текст можно представлять себе как `P = g (K private, C)`. Функция `f` шифрования используется только для шифрования; функция дешифрования `g` используется для дешифрования. Далее мы покажем, что функция `f` нуждается в "лазейке" односторонней функции, чтобы позволить Бобу расшифровывать сообщение, но препятствовать Еве делать то же самое.

Потребность в обеих криптосистемах

Есть очень важный факт, который иногда неправильно истолковывается. Появление асимметрично-ключевой криптографии (открытый ключ доступа) не устраняет потребность в криптографии с

симметричными ключами (секретный ключ). Причина в том, что криптография с асимметричными ключами использует *математические функции* для шифрования и дешифрования намного медленнее, чем криптография с симметричными ключами. Для шифровки больших сообщений криптография с *симметричными ключами* необходима. С другой стороны, скорость криптографии с *симметричными ключами* не устраняет потребность в асимметрично-ключевой криптографии. Асимметрично-ключевая криптография необходима для установления подлинности цифровых подписей и работы станций по рассылке ключей засекречивания. Это означает способность системы использовать все аспекты безопасности. Сегодня мы нуждаемся в обеих системах криптографии. Одна *криптосистема* дополняет другую.

"Лазейка" в односторонней функции

Главная идея асимметрично-ключевой криптографии — понятие "лазейки" в *односторонней функции*.

Функции

Хотя понятие функции знакомо из математики, мы дадим неофициальное определение здесь. Функция — правило, по которому связывают (отображают) один элемент во множестве A, называемый доменом, и один элемент во множестве B, называемый диапазоном, как показано на рис. 14.3.

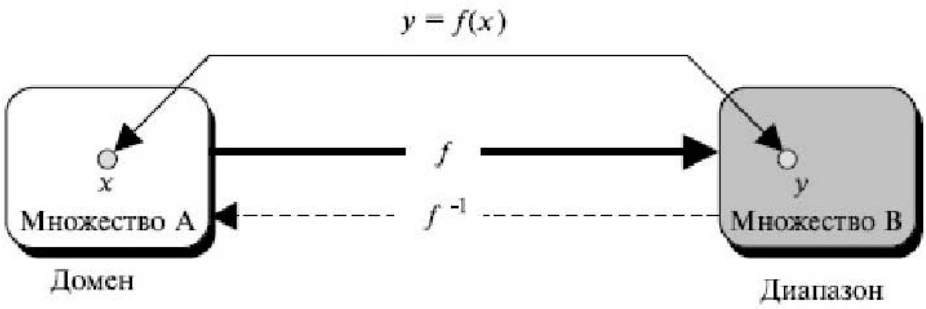


Рис. 14.3. Функция отображения домена в диапазон

Обратимая функция — функция, которая связывает каждый элемент в диапазоне с точно одним элементом в домене.

Односторонняя функция (*OWF* — One Way Function) — функция, которая обладает следующими двумя свойствами:

1. `f` вычисляется просто. Другими словами, при данном `x` может быть легко вычислен `y = f (x)`.
2. `f -1` вычисляется трудно. Другими словами, при данном `y`, вычислить `x= f ~1 (y)` неосуществимо.

"Лазейка" в односторонней функции

"Лазейка" в односторонней функции (TOWF — Trapdoor One Way) — *односторонняя функция* с третьим свойством:

3. При данном `y` и ловушке (секретной) `x` может быть легко вычислен.

Пример 14.1

Когда `n` является большим, $n = p \times q$ — *односторонняя функция*. Обратите внимание, что в этой функции `x` — *кортеж* `(p, q)` двух простых чисел, а `y` в данном случае — это `n`. При заданных `p` и `q` всегда просто вычислить `n`. При данном `n` очень трудно вычислить `p` и `q`. Это — проблема разложения на множители, которую мы рассматривали в лекциях 12-13. В этом случае для нахождения функции `f -1` нет решения с полиномиальным временем.

Пример 14.2

Когда `n` является большим, функция $y = x^k \bmod n$ — "лазейка" в *односторонней функции*. При заданных `x`, `k` и `n` просто вычислить `y`, используя алгоритм быстрого возведения в степень, который мы обсуждали в лекциях 12-13. При заданных `y`, `k` и `n` очень трудно вычислить `x`. Это — проблема дискретного логарифма, которую мы обсуждали в лекциях 12-13. В этом случае нет решения с полиномиальным временем для функции `f -1`. Однако если мы знаем

"лазейку" и k′, такое, что $k \times k' = 1 \bmod \varphi(n)$, мы можем использовать x = y^k mod n, чтобы найти x. Это — известный *алгоритм (RSA* — Riverst-Shamir-Adelman), который будет рассмотрен позже в этой лекции.

Ранцевая криптосистема

Первая блестящая идея относительно криптографии открытого ключа доступа принадлежит Меркелю и Хеллману — она изложена в их ранцевой криптосистеме. Хотя эта система ненадежна в соответствии с сегодняшними стандартами, главная ее идея дает возможность понять современные криптосистемы с открытым ключом, которые будут рассматриваться позже в этой лекции.

Если нам говорят, какие элементы заранее определенного множества чисел мы имеем, то можно легко вычислить сумму чисел. Если нам говорят сумму, то трудно сказать, какие элементы "находятся в ранце".

Определение

Предположим, что нам даны два k *-кортежа*, a = [a_1,a_2, …,a_k] и x = [x_1,x_2, …,x_k]. Первый *кортеж* — заранее определенное множество; второй *кортеж*, в котором x равен только 0 или 1, определяет, какие элементы a должны быть отброшены в ранце. Сумма элементов в ранце равна

s = knapsackSum (a, x) = x_1 a_1 + x_2 a_2 + • • • + $x_k a_k$

По данным a и x просто вычислить s. Однако по данному s трудно найти x. Другими словами, s = knapsackSum (x, a) вычисляется просто, но x= inv_knapmi (s, a) труден. Функция knapsackSum — *односторонняя функция*, если a - общий k *-кортеж*.

Суперувеличение кортежа

Просто вычислить `knapsackSum` и `inv_knapsackSum`, если - `k` - *кортеж* суперувеличивается. В суперувеличивающемся кортеже $a \geqslant a_1 + a_2 + \bullet\bullet\bullet + a_{i-1}$. Другими словами, каждый элемент (кроме a_1) больше или равен сумме всех предыдущих элементов. В этом случае мы вычисляем `knapsackSum` и `inv_knapsackSum`, как показано в Алгоритме 14.1. Алгоритм `inv_knapsackSum` запускается от наибольшего элемента и продолжает процесс к наименьшему. В каждой итерации он проверяет, находится ли элемент в рюкзаке.

```
knapsackSum (x[1,…,k], a[1,…,k] )
{
s <- 0
for (i=1 to k)
{
s <- s + a_i x x_i
}
return x
}

Inv_knapsackSum (s, a[1,…,k])
{
     for (i=k down to 1)
    {
       if s >= a_j
      {
            x_i <- 1
            s <- s - a_i
       }
       else x_i <- 0
    }
     return x[1,…,k]
}
```

Пример 14.1. knapsacksum и inv_knapsackSum для суперувеличивающегося k-кортежа

Пример 14.3

Как очень тривиальный пример, предположим, что даны `a = [17, 25, 46, 94, 201, 400]` и `s = 272`. Таблица 14.1 показывает, как найти *кортеж*, используя процедуру `inv_knapsackSum` в алгоритме 14.1.

В этом случае `x = [0, 1, 1,0, 1,0]`, — это означает, что в рюкзаке находятся `25, 46` и `201`.

Таблица 14.1. Значения i, a и x в примере 14.3

i	a_i	s	s $\geq$	x_i	s $\leftarrow$ s - a_i x x_i
6	400	272	false	x_6=0	272
5	201	272	true	x_5=1	71
4	94	71	false	x_4=0	71
3	46	71	true	x_3=1	25
2	25	25	true	x_2=1	0
1	17	0	false	x_1=0	0

Секретная связь с использованием ранца

Посмотрим, как Алиса может передать секретное сообщение Бобу, использующему ранцевую криптосистему. Идея показана на рис. 14.4.

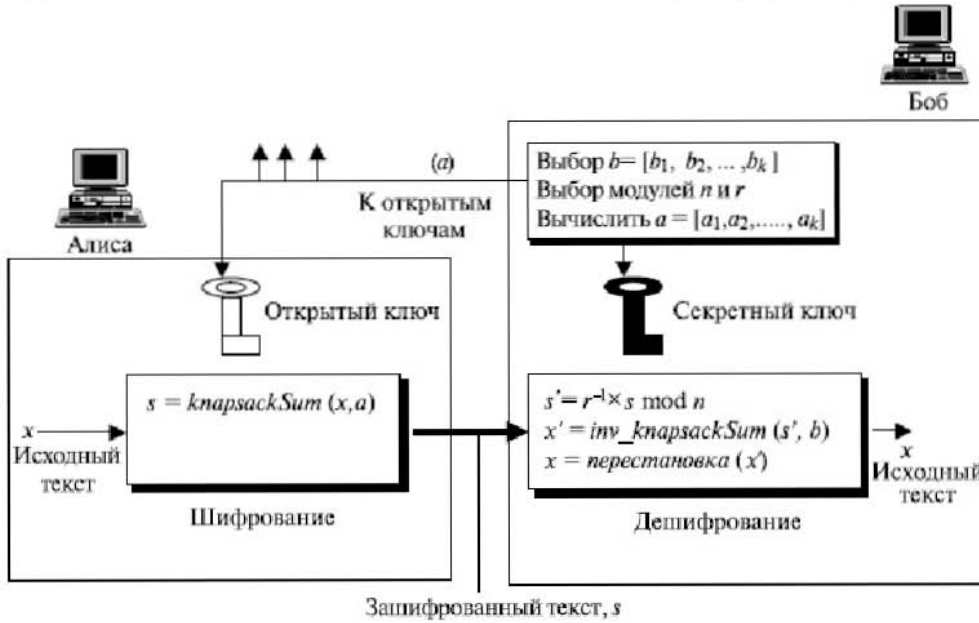


Рис. 14.4. Секретная связь с использованием ранцевой криптосистемы

Генерация ключей

Этот процесс:

- Создает суперувеличивающийся `k` -*кортеж* `b = [b1, b2,..., bk]`.
- Выбирает модуль `n`, такой, что `n > b 1 + b2 + • • • + bk`.
- Выбирает случайное целое число `r`, которое является взаимно простым с `n` и $1 \leqslant r \leqslant n - 1$.
- Создает временный `k` -*кортеж* `t = [t1, t2,..... tk]`, в котором $t_i = r \times b_i \bmod n$.
- Выбирает перестановку `k` -объектов и находит новый *кортеж* `a = liermute(t)`.
- Открытый ключ `k` -*кортежа* — `a`. Секретный ключ — `n, r` и `k` -*кортеж* `b`.

Шифрация

Предположим, что Алисе надо послать сообщение Бобу.

- Алиса преобразует свое сообщение в `k` -*кортеж* `x=[`x_1`, `x_2`,…. `x_k`]`, в котором x_i — не `0` и не `1`. *Кортеж* представляет собой исходный текст.
- Алиса использует `knapsackSum` для вычисления `s` в качестве исходного текста.

Дешифрация

Боб получает зашифрованный текст `s`.

Боб вычисляет $s' = r^{-1} \times s \bmod n$.

Боб переставляет `x'` для того, чтобы найти `x`. *Кортеж* `x` есть восстановленный исходный текст.

Пример 14.4

Это тривиальный (очень легко раскрываемый пример). Он приводится только для того, чтобы показать процедуру.

1. *Генерация ключей*:
    - Боб создает суперувеличивающийся *кортеж* `b = [7, 11, 19, 39, 79, 157, 313]`.
    - Боб выбирает модуль `n = 900` и `r = 37`, и `[4 2 5 3 1 7 6]` как таблицу перестановок.
    - Боб теперь вычисляет *кортеж* `t = [259, 407, 703, 543, 223, 409, 781]`.
    - Боб теперь вычисляет *кортеж* `a` = перестановка `(t)= [543, 407, 223, 703, 259, 781, 409]`.
    - Боб объявляет `a` ; он сохраняет в тайне `n, r` и `b`.
2. Предположим, что Алиса хочет передать единственный символ `"g"` Бобу.
    - Она использует представление ASCII на `7` битов `"g"`,

(1100111) $_2$ и создает *кортеж* x = [1,1,0,0, 1, 1, 1]. Это — исходный текст.

- Алиса вычисляет s = knulisackSum (a, x) = 2399. Это — зашифрованный текст, передаваемый Бобу.

3. Боб может расшифровать зашифрованный текст, s = 2165.
   - а. Боб вычисляет $s' = r^{-1} \times s \bmod n = 2165 \times 37^{-1} \bmod 900 = 527$.
   - Боб вычисляет x′ = inv_knalisackSum (s', b) = [1, 1, 0, 1, 0, 1, 1].
   - Боб вычисляет x = перестановка (x′) = [1, 1, 0, 0, 1, 1, 1]. Он интерпретирует строку (1100111) $_2$ как символ "g".

Лазейка

Вычисление суммы элементов в ранце Алисы — фактически *умножение матрицы*-строки x на матрицу-столбец a. Результат — матрица s 1×1. Матричное умножение: $s = x \times a$, в котором x является матрицей-строкой, а a — матрица-столбец -*односторонняя функция*. По данным s и x Ева не сможет легко найти a. Боб, однако, имеет лазейку. Боб использует его $s' = r^{-1} \times s$ и секретную, суперувеличивающуюся матрицу-столбец b, чтобы найти матрицу-строку x. При этом он применяет процедуру inv_knapsackSum. Перестановка позволяет Бобу найти x по известному x′.

14.2. Криптографическая система RSA

Самый общий алгоритм открытого ключа доступа — криптографическая система RSA, названная по имени его изобретателей Ривеста, Шамира, Эделмана (Rivest, Shamir и Adelman).

Введение

RSA использует два типа ключей — e и d, где e — открытый, а d —

секретный. Предположим, что P — исходный текст и C — зашифрованный текст. Алиса использует $C = P^e \bmod n$, чтобы создать зашифрованный текст C из исходного текста P ; Боб использует $P = C^d \bmod n$, чтобы извлечь исходный текст (файл), переданный Алисой. Модулей n создается очень большое количество с помощью процесса генерации ключей, который мы обсудим позже.

Для шифрования и дешифрования применяют возведение в степень по модулю. Как мы уже обсуждали в лекциях 12-13, при использовании быстрого алгоритма возведение в степень по модулю выполнимо в полиномиальное время. Однако нахождение модульного логарифма так же сложно, как и разложение числа по модулю. Для него нет алгоритма с полиномиальным временем. Это означает, что Алиса может зашифровать сообщение общедоступным ключом (e) в полиномиальное время. Боб также может расшифровать его в полиномиальное время (потому что он знает d). Но Ева не может расшифровать это сообщение, потому что она должна была бы вычислить корень e -той степени из C с использованием модульной арифметики. Рисунок 14.5 показывает идею *RSA*.

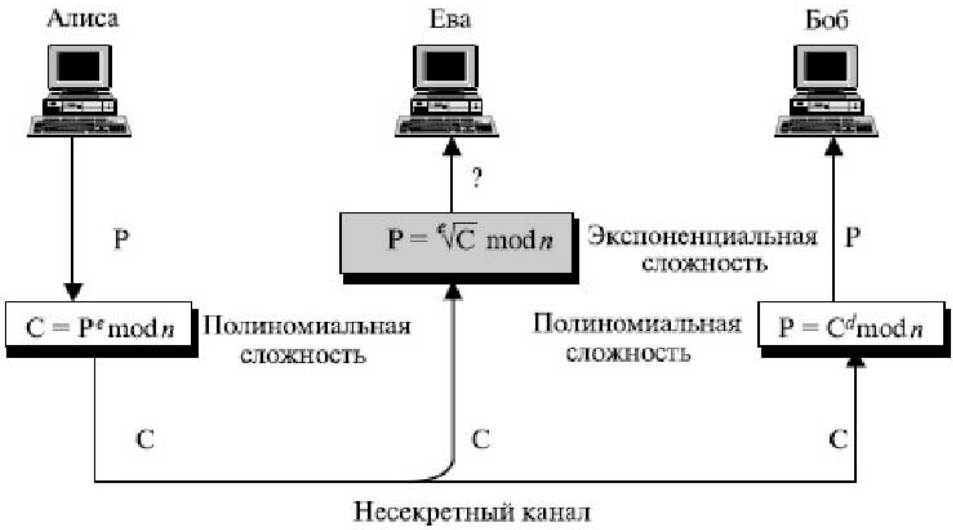


Рис. 14.5. Сложность операций в RSA

Другими словами, Алиса использует *одностороннюю функцию* (возведение в степень по модулю) с лазейкой, известной только Бобу.

Ева не знает лазейку, поэтому не может расшифровать сообщение. Если когда-нибудь найдут полиномиальный алгоритм для модуля вычисления корня `e` -той степени из `n`, то возведение в степень по модулю `n` не будет больше односторонней функцией.

Процедура

Рисунок 14.6 показывает общую идею процедуры, используемой в *RSA*.

RSA использует возведение в степень по модулю для шифрования/дешифрования. Для того чтобы атаковать закрытый текст, Ева должна вычислить $\sqrt[e]{C} \bmod n$

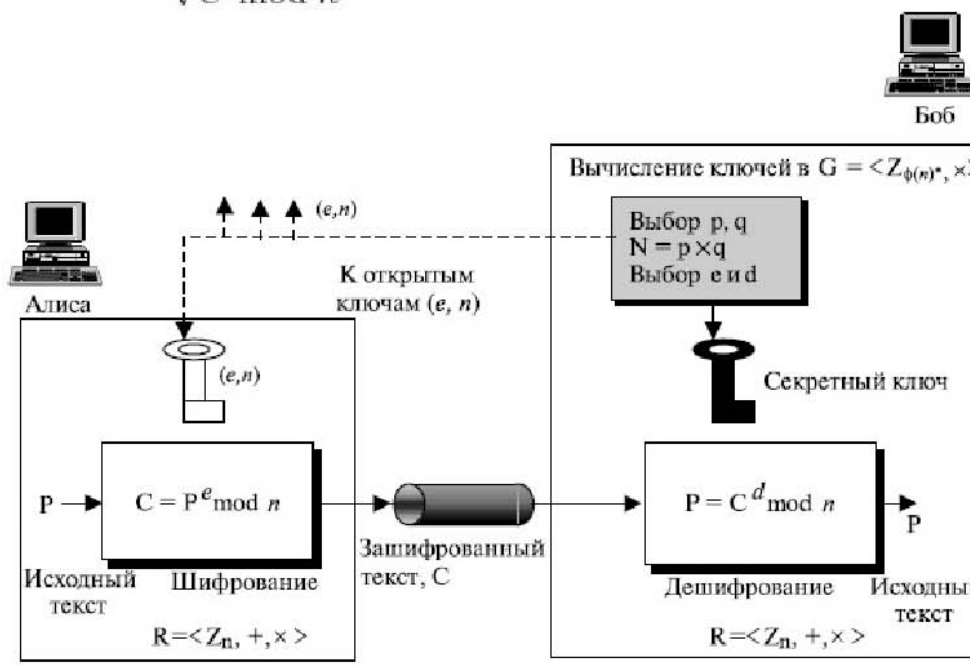


Рис. 14.6. Шифрование, дешифрование и генерация ключей в RSA

Две алгебраические структуры

RSA использует две алгебраических структуры: кольцо и группу.

Кольца шифрования/дешифрования. Шифрование и дешифрование

сделаны с использованием *коммутативного* кольца $R = Z_n, +, \times$ с двумя арифметическими операциями: сложение и умножение. В *RSA* это кольцо общедоступно, потому что модуль n общедоступен. Любой может послать сообщение Бобу, используя это кольцо для шифрования.

Группы генерирования ключей. *RSA* использует мультипликативную группу $G = Z_{\varphi(n)*}, \times$ для генерации ключей. Группа поддерживает только умножение и деление (мультипликативную инверсию), которые необходимы для того, чтобы создать открытые и секретные ключи. Эту группу надо скрыть, потому что ее модуль $\varphi(n)$ является секретным. Мы увидим, что если Ева найдет этот модуль, она сможет легко атаковать криптографическую систему.

RSA использует две алгебраических структуры: открытое кольцо R = < Z $_n$, +, x > и секретную группу G = < Z ϕ (n)* , x >.

Генерация ключей

Боб использует шаги, показанные в алгоритме 14.2, чтобы создать свои открытый и секретный ключи. После генерации ключей Боб объявляет *кортеж* (e, n) как свой открытый ключ доступа: Боб сохраняет d как свой секретный ключ. Боб может отказаться от p, q и $\varphi(n)$; они не могут изменить его секретный ключ, не изменяя модуль. Для безопасности рекомендуется размер для каждого простого p или q — 512 бит (почти 154 десятичные цифры). Это определяет размер модуля, n 1024 бита (309 цифр).

RSA Key_Generation (RSA - генерация ключа)
{
Выбрать два больших простых p and q, таких, что $p \neq q$.
$n \leftarrow p \times q$
$\varphi(n) \leftarrow (p-1) \times (q-1)$
Выбрать e, такое, что $1 < e < \varphi(n)$ и e is - взаимно простое с $\varphi(n)$
$d \leftarrow e^{-1} \mod \varphi(n)$ // d - это инверсия e по модулю $\varphi(n)$
Открытый ключ $\leftarrow (e, n)$ // Объявляется открытым
Секретный ключ $\leftarrow d$ // Сохраняется в секрете
return Public_key and Private_key // Возврат открытого и секретного ключей
}

Пример 14.2. RSA-генерация ключей

В RSA кортеж (e, n) — открытый ключ доступа; целое число d — секретный ключ.

Шифрование

Передать сообщение Бобу может любой, используя его открытый ключ доступа. Шифрование в *RSA* может быть выполнено с использованием алгоритма с полиномиальной сложностью по времени, как показано в алгоритме 14.3. Быстрый алгоритм возведения в степень был рассмотрен в лекциях 12-13. Размер исходного текста должен быть меньше чем n ; если размер исходного текста больше, то он должен быть разделен на блоки.

RSA_Encryption (P, e, n) // P — исходный текст в Z_n и P < n
{
C <- Fast_Exponentiation (P, e, n) //Вычисление (P^e mod n)
return C
}

Пример 14.3. Шифрование RSA

Дешифрование

Чтобы расшифровать сообщение зашифрованного текста, которое Боб получил в *RSA*, он может использовать алгоритм 14.4. Это можно выполнить, используя алгоритм с полиномиальной сложностью по времени, если размер зашифрованного текста меньше, чем n.

```
RSA_Decryption (C, d, n)        //C — зашифрованный текст в Z_n
{
  P <- Fast_Exponentiation (C, d, n)        // Вычисление (C^d mod n)

  return P
}
```

Пример 14.4. Дешифрование RSA

В RSA p и q должны быть по крайней мере 512 битов; n должны быть по крайней мере 1024 бит.

Доказательство RSA

Используя вторую версию теоремы Эйлера, которая обсуждалась в лекциях 12-13, мы можем доказать, что шифрование и дешифрование инверсны друг другу.

Если $n = p \times q < n$, и k - целое число, тогда $a^{k \times \varphi(n)+1} \equiv a (mod\ n)$.

Предположим, что исходный текст, восстановленный Бобом, есть P_1. Докажем, что он эквивалентен P.

$$P_1 = C^d \mod n = (P^e \mod n) \mod n = P^{ed} \mod n$$
$$ed = k\varphi(n) + 1 \qquad // \ d \text{ и } e \text{ инверсны по модулю } \varphi(n)$$
$$P_1 = P^{ed} \mod n \rightarrow P_1 = P^{k\varphi(n)+1} \mod n$$
$$P_1 = P^{k\varphi(n)+1} \mod n = P \mod n \qquad // \text{ Теорема Эйлера (вторая версия)}$$

Некоторые тривиальные примеры

Рассмотрим некоторые тривиальные (ненадежные) примеры процедуры *RSA*. Критерии, которые делают систему RSA безопасной, будут

обсуждены в более поздних разделах.

Пример 14.5

Боб выбирает 7 и 11 как p и q и вычисляет $n = 7 \times 11 = 77$. Значение $\varphi(n) = (7-1)(11-1)$ или 60. Теперь он выбирает два ключа, e и d, из Z_{60}^*. Если он выбирает e = 13, то d = 37. Обратите внимание, что $e \times d \bmod 60 = 1$ (они инверсны друг другу). Теперь предположим, что Алиса хочет передать исходный текст 5 Бобу. Она использует общедоступный ключ 13, чтобы зашифровать 5.

Исходный текст:5 C = 5^{13} = 26 mod 77 Зашифрованный текст: 2

Боб получает зашифрованный текст 26 и использует секретный ключ 37, чтобы расшифровать зашифрованный текст.

Зашифрованный текст: 26 P = от 26^{37} до 5 mod 77 Исходный тек

Переданный Алисой текст получен Бобом как исходный текст 5.

Пример 14.6

Теперь предположим, что другой человек, Джон, хочет передать сообщение Бобу. Джон может использовать открытый ключ доступа, объявленный Бобом (вероятно, на его сайте), - 13 ; исходный текст Джона — 63. Джон делает следующие вычисления:

Исходный текст: 63 C = 63^{13} = 28 mod 77 Зашифрованный текст:

Боб получает зашифрованный текст 28 и использует свой секретный ключ 37, чтобы расшифровать зашифрованный текст.

Зашифрованный текст: 28 P = 28^{37} = 63 mod 77 Исходный текст: 63

Пример 14.7

Дженнифер создает пару ключей для себя. Она выбирает p = 397 и q = 401. Она вычисляет $n = 397 \times 401 = 159197$. Затем она

вычисляет $\varphi(n) = 396 \times 400 = 158400$. Затем она выбирает `e = 343` и `d = 12007`. Покажите, как Тэд может передать сообщение `"No"` Дженнифер, если он знает `e` и `n`.

Решение

Предположим, что Тэд хочет передать сообщение `"No"` Дженнифер. Он изменяет каждый символ на число (от `00` до `25`), сопоставляет каждой букве число, содержащее две цифры. Затем он связывает два *кодированных символа* и получает четырехзначное число. Исходный текст — `1314`. Затем Тэд использует `e` и `n`, чтобы зашифровать сообщение. Зашифрованный текст $1314^{343} = 33677 \bmod 159197$. Дженнифер получает сообщение `33677` и использует `d` ключ дешифрования, чтобы расшифровать это сообщение: $33677^{12007} = 1314 \bmod 159197$. Затем Дженнифер расшифровывает `1314` как сообщение `"No"`. Рисунок 14.7 показывает этот процесс.

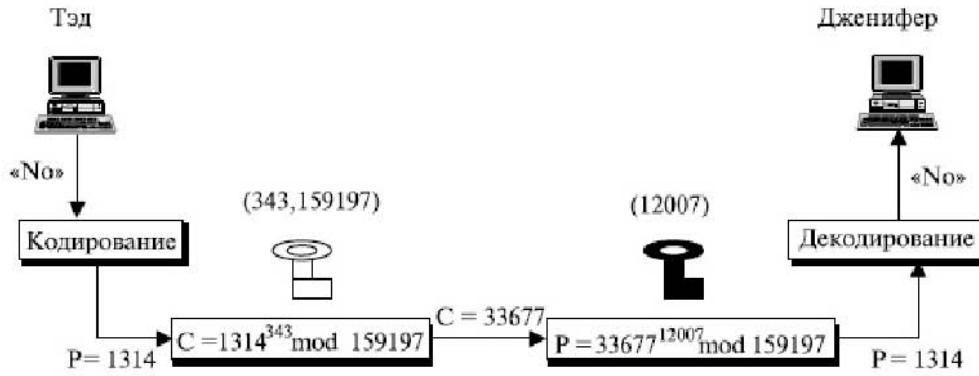


Рис. 14.7. Шифрование и дешифрование в примере 14.7

Атаки RSA

До настоящего момента не было обнаружено никаких разрушительных атак RSA. Несколько атак были предсказаны. Они основаны на слабом исходном тексте, слабом выборе параметра или несоответствующей реализации. Рисунок 14.8 показывает категории потенциальных атак.

Рис. 14.8. Диаграмма возможных атак на RSA

Атака разложения на множители

Безопасность RSA базируется на следующей идее: модуль настолько большой, что разложение на множители в разумное время неосуществимо. Боб выбирает `p` и `q` и вычисляет $n = p \times q$. Число `n` общедоступно, `p` и `q` являются секретными. Если Ева сможет разложить на множители `n` и получить `p` и `q`, то она может вычислить $\varphi(n) = (p-1)(q-1)$. Затем Ева тогда может вычислить $d = e^{-1} \bmod \varphi(n)$, потому что `e` общедоступен. Секретный ключ `d` — лазейка, которую Ева может использовать, чтобы расшифровать зашифрованное сообщение.

Как мы узнали в лекциях 12-13, есть много алгоритмов разложения на множители, но ни один из них не может найти сомножители большого целого числа с полиномиальной сложностью времени. Для того чтобы обеспечить безопасность, *RSA* требует, чтобы `n` был больше чем `300` десятичных цифр. Это означает, что модуль должен быть по крайней

мере 1024 бита. Даже при использовании мощнейшего и самого быстрого компьютера, доступного на сегодня, разложение на множители целого числа такого размера требует неосуществимо большого времени. Это означает, что *RSA* безопасен, пока не будет найден эффективный алгоритм разложения на множители.

Атака с выборкой зашифрованного текста

Потенциальная атака RSA базируется на мультипликативном свойстве *RSA*. Предположим, Алиса создает зашифрованный текст $C = P^e \bmod n$ и передает C Бобу. Также предположим, что Боб расшифрует произвольный зашифрованный текст для Евы – C_1, отличный от C. Ева перехватывает C и использует следующие шаги, чтобы найти P:

а. Ева выбирает случайное целое число X в Z_{n*}.

б. Ева вычисляет $Y = C \times X^e \bmod n$.

в. Ева передает Y Бобу для дешифрования и получает $Z = Y^d \bmod n$; это шаг атаки выборкой зашифрованного текста.

г. Ева может легко найти P, потому что

$$Z = Y^d \bmod n = (C \times X^e)^d \bmod n = (C^d \times X^{ed}) \bmod n = (C^d \times X) \bmod n = (P \times$$
$$Z = (P \times X) \bmod n \rightarrow P = Z \times X^{-1} \bmod n$$

Ева использует расширенный евклидов алгоритм для того, чтобы найти мультипликативную инверсию X, и в конечном счете значение P.

Атаки на показатель степени шифрования

Чтобы уменьшить время шифрования, можно попытаться использовать короткий ключ шифрования — малое значение числа e, например, значение для e, такое как $e = 3$ (второе простое число). Однако есть некоторые потенциальные атаки на показатель при его малом значении степени шифрования, которые мы здесь кратко обсуждаем. Эти атаки вообще не кончаются вскрытием системы, но они все-таки должны

быть предотвращены. Для того чтобы сорвать эти виды атак, рекомендуется использовать $e = 2^{16} + 1 = 65537$ (или простое число, близкое к этому значению).

Атака теоремы Куперсмита (Coppersmith) может быть главной для атаки малого показателя степени на ключ шифрования. Основное положение этой теоремы: для полинома $f(x)$ степени e по модулю n, чтобы найти корни, если один из корней является меньшим чем $n^{1/e}$, можно использовать алгоритм сложности, $\log n$. Эта теорема может быть применена к *RSA-криптосистеме* $C = f(P) = P^e \bmod n$. Если $e = 3$ и известны хотя бы две трети битов в исходном тексте P, алгоритм может найти все биты в исходном тексте.

Атака широковещательной передачи может быть начата, если один объект передает одно и то же сообщение группе получателей с тем же самым ключом шифрования. Например, предположим следующий сценарий: Алиса хочет передать одно и то же сообщение трем получателям с тем же самым общедоступным ключом $e = 3$ и модулями n_1, n_2 и n_3.

$$C_1 = P^3 \bmod n_1$$
$$C_2 = P^3 \bmod n_2$$
$$C_3 = P^3 \bmod n_3$$

Применяя китайскую теорему об остатках к этим трем уравнениям, Ева может найти уравнение формы $C' = P^3 \bmod n_1 n_2 n_3$. Это означает, что $P^3 < n_1 n_2 n_3$ и что $C' = P^3$ решается с помощью обычной арифметики (не модульной). Ева может найти значение $C' = P^{1/3}$.

Атака связанных между собой сообщений была обнаружена Франклином Рейтером (Franklin Reiter). Она может быть кратко описана следующим образом. Алиса зашифровала два исходных текста, P_1 и P_2, с помощью $e = 3$ и передает C_1 и C_2 Бобу. Если P_1 связан с P_2 *линейной функцией*, то Ева может восстановить P_1 и P_2 в выполнимое время вычисления.

Атака короткого списка, обнаруженная Куперсмитом, может быть кратко описана следующим образом. Алиса имеет сообщение M для передачи Бобу. Она записывает сообщение и зашифровывает его как сообщение r_1, а результат записывает как C_1 и передает C_1 (Бобу). Ева перехватывает C_1 и удаляет его. Боб сообщает Алисе, что он не получил сообщение, так что Алиса заполняет сообщение, снова зашифровывает как сообщение r_2 и передает это Бобу. Ева также перехватывает и это сообщение. Ева теперь имеет C_1 и C_2, и она знает, что оба зашифрованных текста принадлежат одному и тому же исходному тексту. Куперсмит доказал, что если r_1 и r_2 короткие, то Ева способна восстановить первоначальное сообщение M.

Атаки показателя степени дешифрации

Две формы атак могут быть проведены на показатель степени дешифрации: атака раскрытой степени дешифрации и атака малого показателя степени дешифровации. Они обсуждаются ниже.

Атака раскрытого показателя степени дешифрации. Очевидно, что если Ева может найти показатель степени дешифрации, d, она сможет расшифровать текущее зашифрованное сообщение. Однако на этом атака не останавливается. Если Ева знает значение d, она может использовать вероятностный алгоритм (не обсуждаемый здесь) к числу n и найти значения p и q. Следовательно, если Боб изменит только угрожающий безопасности показатель степени дешифрования, но сохранит тот же самый модуль n, Ева сможет расшифровать будущие сообщения, потому что она сможет разложить на множители n. Поэтому если Боб узнает, что показатель степени скомпрометирован, он должен выбрать новое значение для p и q, вычислить n и создать полностью новые секретный и открытый ключи доступа.

В RSA, если показатель степени d скомпрометирован, тогда p, q, n, e и d должны быть сгенерированы заново.

Атака малого значения показателя степени дешифрации. Боб может подумать, что использование малого значения степени секретного ключа d приводит к более быстрой работе алгоритма дешифрации.

Винер показал, что в случае $d < 1/3n^{1/4}$ возможен специальный тип атаки, основанной на цепной дроби, — тема, которая рассматривается в теории чисел. Этот тип атаки может подвергнуть риску безопасность RSA. Для того чтобы это произошло, должно выполняться условие, что $q < p < 2q$; если эти два условия существуют, Ева может разложить n на сомножители в полиномиальное время.

В RSA рекомендовано, что d должно иметь величину $d > 1/3\ n^{1/4}$, чтобы предотвратить атаку малого значения ключа дешифрации.

Атаки исходного текста

Исходный текст и зашифрованный текст в *RSA* — это перестановки друг друга, потому что это целые числа в том же самом интервале (от 0 до $n - 1$). Другими словами, Ева уже знает кое-что об исходном тексте. Эти характеристики могут позволить некоторые атаки исходного текста. Три атаки были уже упомянуты в литературе: атака короткого сообщения, атака циклического повторения и явная атака.

Атака короткого сообщения. В атаке короткого сообщения, если Ева знает множество возможных исходных текстов, то ей известна еще одна информация и дополнительный факт, что зашифрованный текст — перестановка исходного текста. Ева может зашифровать все возможные сообщения, пока результат не будет совпадать с перехваченным зашифрованным текстом. Например, если известно, что Алиса посылает число с четырьмя цифрами Бобу, Ева может легко испытать числа исходного текста 0000 к 9999, чтобы найти исходный текст. По этой причине короткие сообщения должны быть дополнены случайными битами в начале и конце, чтобы сорвать этот тип атаки. Настоятельно рекомендуется заполнять исходный текст случайными битами прежде начала шифрования. Здесь используется метод, называемый OAEP, который будет позже обсужден в этой лекции.

Атака циклического повторения построена на факте, что если переставлять зашифрованный текст (перестановка исходного текста), то непрерывное шифрование зашифрованного текста в конечном счете кончится исходным текстом. Другими словами, если Ева непрерывно шифрует перехваченный зашифрованный текст C, она в итоге получит

исходный текст. Однако сама Ева не знает, каков исходный текст, так что ей неизвестно, когда пора остановиться. Она должна пройти один шаг далее. Когда она получает зашифрованный текст C снова, она возвращается на один шаг, чтобы найти исходный текст.

Перехваченный зашифрованный текст C
$C_1 = C^e \bmod n$
$C_2 = C_1^e \bmod n$
……
$C_k = C_{k-1}^e \bmod n$ ->, если $C_k = C$, останов: исходный текст - $P = C_{k-1}$

Может ли это быть серьезной атакой на криптосистему *RSA*? Показано, что сложность алгоритма эквивалентна сложности разложения на множители n. Другими словами, нет никакого эффективного алгоритма, который может завершить эту атаку в полиномиальное время, если n является большим.

Явная атака сообщения. Другая атака, которая базируется на отношениях перестановки между исходным текстом и зашифрованным текстом, — явная атака сообщения. Явное сообщение — сообщение, которое зашифровано само в себя (не может быть скрыто). Было доказано, что есть всегда некоторые сообщения, которые шифруются сами в себя. Поскольку ключ шифрования обычно нечетен, имеются некоторые исходные тексты, которые зашифрованы сами в себя, такие как P = 0 и P = 1. Но если ключ шифровки выбран тщательно, число их незначительно. Программа шифровки может всегда проверить, является ли вычисленный зашифрованный текст таким же, как исходный текст, и отклонить $P^{e}B$ исходный текст перед передачей зашифрованного текста.

Атаки модуля

Главной атакой *RSA* является атака разложения на множители. Ее можно рассматривать как атаку малого модуля. Однако поскольку мы уже обсудили эту атаку, мы концентрируемся на другой атаке модуля: общей атаке модуля.

Общая атака модуля. Она может быть начата, если сообщество C^{d_B} использует общий модуль, n. Например, люди в сообществе могли бы позволить третьей стороне, которой они доверяют, выбирать p и q, вычислять n и $\varphi(n)$ и создать пару образцов (e_i, d_i) для каждого объекта. Теперь предположим, что Алиса должна передать сообщение Бобу. Зашифрованный текст Бобу — это $C = P^{e_B} \bmod n$ Боб использует свой секретный ключ, d_B, чтобы расшифровывать сообщение: $P = C^{d_B} \bmod n$. Проблема в том, что Ева может также расшифровать сообщение, если она — член сообщества и ей была назначена пара образцов (e_E и d_E), как мы узнали в разделе "атака малого значения ключа дешифрации". Используя свои собственные ключи (e_E и d_E), Ева может начать вероятностную атаку на сомножители n и найти d_B Боба. Чтобы сорвать этот тип атаки, модуль не должен быть в совместном пользовании. Каждый объект должен вычислить свой собственный модуль.

Атаки реализации

Предыдущие атаки базировались на основной структуре RSA. Как показал Дэн Бонех (Dan Boneh), есть несколько атак реализации RSA. Мы приведем две из них: атака анализом времени и атака мощности.

Атака анализом времени (Timing attack). Пауль Кочер (Paul Kocher) демонстрировал атаку только зашифрованного текста, называемую атака анализом времени. Атака основана на быстром алгоритме с показательным временем, который рассмотрен в лекциях 12-13. Алгоритм использует только возведение во вторую степень, если соответствующий бит в секретном показателе степени d есть 0 ; он используется и при возведении во вторую степень и умножении, если соответствующий бит — 1. Другими словами, синхронизация требует сделать каждую итерацию более длинной, если соответствующий бит — 1. Эта разность синхронизации позволяет Еве находить значение битов в d, один за другим.

Предположим, что Ева перехватила большое количество зашифрованных текстов от C_1 до C_m. Также предположим, что Ева

наблюдала, какое количество времени требуется для Боба, чтобы расшифровать каждый зашифрованный текст, от T_1 до T_2. Ева знает, сколько времени требуется для основных аппаратных средств, чтобы выполнить операцию умножения от t_1 до t_m., где t_1 — время, требуемое для выполнения умножения.

Результат операции умножения = Результат $\times C_i \bmod n$. Ева может использовать алгоритм 14.5, который является упрощенной версией алгоритма, используемого практически для вычисления всех бит в d (d_0 до d_{k-1}).

Алгоритм устанавливает начальное значение $d_0 = 1$ (потому что d должен быть нечетным) и вычисляет новые значения для $T'_i s$ (время дешифрования относится к d_1 до d_{k-1}). Алгоритм затем предполагает, что следующий бит — это 1, и находит несколько значений D_1 до D_2, основываясь на этом предположении.

```
RSA_Timing_Attack([T1…Tm])
{
 d0 <- 1        // Потому что d — нечетное
 Вычислить [t1…tm]
 [T1…Tm] <- [T1…Tm] - [t1…tm]
 for (j from 1 to k-1)
  {                    //Обновление Ti для следующего бита

  Пересчитать [t1…tm]
        //Пересчет ti  в предположении, что следующий бит — это 1

  [D1…Dm] <- [T1…Tm] -[t1…tm]
  var <- variance ([D1…Dm]) – variance ([T1…Tm])
  if (var > 0) dj <- 1  else dj <- 0

  [T1…Tm] <- [T1…Tm] - dj x [t1…tm]          //Обновление Ti для следующ

  }
}
```

Пример 14.5. Атака синхронизации

Если принятое предположение верно, то каждый D_i является вероятно меньшим, чем соответствующее время передачи T_i. Однако алгоритм использует дисперсию (или другие критерии корреляции), чтобы рассмотреть все варианты D_i и T_i. Если разность *дисперсии* положительная, алгоритм принимает предположение, что следующий бит равен 1 в противном случае предполагает, что следующий бит — 0. Алгоритм тогда вычисляет новые T_i, используя для этого оставшиеся биты.

Есть два метода сорвать атаку анализом времени:

1. добавить случайные задержки к возведению в степень, чтобы каждое возведение в степень занимало одно и то же время;

2. Ривест рекомендовал "ослепление". По этой идее зашифрованный текст умножается на случайное число перед дешифрованием. Процедура содержит следующие шаги:

a. Выбрать секретное случайное число r между 1 и (n - 1).

b. Вычислить $C_1 = C \times r^e \bmod n$.

c. Вычислить $P_1 = C_1^d \bmod n$.

d. Вычислить $P = P_1 \times r^{-1} \bmod n$.

Атака анализом мощности подобна атаке анализом времени. Было показано, что если Ева может точно измерить мощность, использованную в течение дешифрования, она может начать атаку анализа мощности на основании принципов, рассмотренных для атаки анализом времени. *Итеративное* умножение и возведение в квадрат потребляют больше мощности, чем только *итеративное* возведение в квадрат. Та же самая группа методов, которая предотвращает атаки анализом времени, может сорвать атаки анализа мощности.

Рекомендации

Следующие рекомендации основаны на теоретических и экспериментальных результатах.

1. Число битов для `n` должно быть, по крайней мере, `1024`. Это означает, что `n` должно быть приблизительно 2^{1024}, или `309` десятичных цифр.
2. Два простых числа `p` и `q` должны каждый быть по крайней мере `512` битов. Это означает, что `p` и `q` должны быть приблизительно 2^{512} или `154` десятичными цифрами.
3. Значения `p` и `q` не должен быть очень близки друг к другу.
4. `p - 1` и `q - 1` должны иметь по крайней мере один большой простой сомножитель.
5. Отношение `p/q` не должно быть близко к *рациональному числу* с маленьким *числителем* или знаменателем.
6. Модуль `n` не должен использоваться совместно.
7. Значение `e` должно быть $2^{16} + 1$ или целым числом, близким к этому значению.
8. Если произошла утечка частного ключа `d`, Боб должен немедленно изменить `n` так же, как `e` и `d`. Было доказано, что знание `n` и одной пары `(e, d)` может привести к открытию других пар того же самого модуля.
9. Сообщения должны быть дополнены, используя OAEP, который рассматривается далее.

Оптимальное асимметричное дополнение шифрования (OAEP — Optimal Assimetric Encryption Padding)

Как мы упоминали ранее, короткое сообщение в *RSA* делает зашифрованный текст уязвимым к атакам короткого сообщения. Там же показано, что простое добавление фиктивных данных (дополнение) к сообщению затрудняет работу Евы, но, приложив дополнительные усилия, она может все еще атаковать зашифрованный текст. Решение, предложенное группой *RSA* и некоторыми другими разработчиками, состоит в том, чтобы применить процедуру, названную оптимальным асимметричным дополнением шифрования (OAEP). Рисунок 14.9 показывает простую версию этой процедуры; реализация может

использовать более сложную версию.

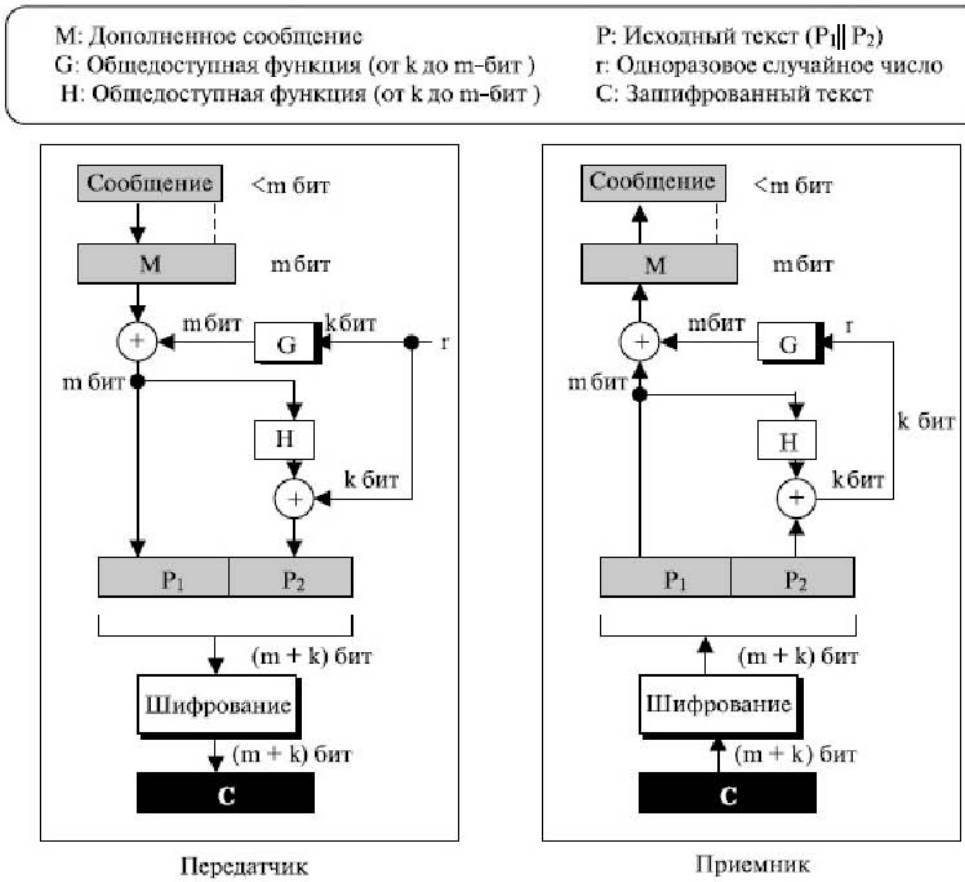


Рис. 14.9. Оптимальное асимметричное дополнение шифрования (OAEP)

Идея, показанная на рисунке 14.9, — это то, что `P = P1 || P2`, где `P1` — замаскированная версия дополненного сообщения, `M`; `P2` передается, чтобы позволить Бобу найти маску.

Шифрование. Ниже показаны шаги процесса шифрования.

1. Алиса дополняет сообщение, чтобы сделать его `m` -битовым. Мы обозначим его `M`.
2. Алиса выбирает случайное число `r` из `k` бит. Обратите внимание, что `r` применяется только однажды и затем уничтожается.

3. Алиса использует общедоступную *одностороннюю функцию* G, которая принимает целое r -битовое число, и создает m - разрядное целое число (m — размера M, и r <m). Это — маска.
4. Алиса применяет маску, G (r), чтобы создать первую часть исходного текста $P_1 = M \oplus G(r)$ является замаскированным сообщением.
5. Алиса создает вторую часть исходного текста $P_2 = H(P_1) \oplus r$. Функция H — другая общедоступная функция, которая принимает m -битовые входные сообщения и создает k -битовые выходные сообщения. Эта функция может быть криптографической хэш-функцией ю P_2 используется для того, чтобы дать возможность Бобу снова создать маску после дешифрации.
6. Алиса создает $C = P^e = (P_1 \,||\, P_2)^e$ и передает C Бобу.

Дешифрование. Следующие шаги показывают процесс дешифрования:

1. Боб создает $P = C^d = (P_1 \,||\, P_2)$.
2. Боб сначала обновляет значение r, используя $H(P_1) \oplus P_2 = H(P_1) \oplus H(P_2) \oplus r = r$.
3. Боб применяет $G(r) \oplus P = G(r) \oplus G(r) \oplus M = M$, чтобы обновить значение дополненного сообщения.
4. После удаления дополнения M, Боб находит первоначальное сообщение.

Ошибка в передаче

Если хотя бы один бит в течение передачи принят с ошибкой, текст, зашифрованный *RSA*, будет принят неправильно. Если полученный зашифрованный текст отличается от переданного, приемник не может определить первоначальный исходный текст. Исходный текст, вычисленный на стороне приемника, может очень отличаться от передаваемого передатчиком. *Среда передачи* должна быть освобожденной от ошибок за счет добавления избыточных бит или обнаружения и исправления ошибки в зашифрованном тексте.

Пример 14.8

Вот — реальный пример. Мы выбираем 512 -битовые p и q, вычисляем n и $\varphi(n)$, затем выбираем e и испытываем, что оно взаимно простое с $\varphi(n)$. Затем мы вычисляем d. Наконец, мы показываем результат шифрования и дешифрования. Целое число p — это число со 159 цифрами.

P = 96130345313583504574191581280615427909309845594996215822585
79647940455056470638491257160180347503120986666064924201918085
06674210960633542199266612 09

Целое число q содержит 160 цифр.

q = 12060191957231446918276794204450896001555925054637033936065
79832173148214848376465921538945320917522527322683010712069566
60251388714552496900035966004561 7

Модуль $n = p \times q$. Это число имеет 309 цифр.

n = 11593504173967614968892509864615887523771457375454144775487
7614788540S32635081727687881596832516846884930062548576411125
241455233918.29271625076567727274600970827141277304349605005 5
72745666280600999240371029914244722922157727985317270338393811
692684137 32762200096667667183183108837342082344437095 3

$\varphi(n) = (p-1)(q-1)$ имеет 309 цифр.

$\varphi(n) = 11593504173967614968892509864615887523771457375454144775485526137614788540832635081727687881596832516846884930062548576411125010624145523391829271625076567510542336084929167520344826279881175547876570139234444057169895817281960982263610754672118646121713591073586406140088851702653772772644673410662438576641 28$

Боб выбирает e = 35535 (идеально 65537), и испытание на простое число показывает, что это число и $\varphi(n)$ — взаимно простые числа. Затем Боб находит инверсию $e \bmod \varphi(n)$ — это обозначается

d.

e = 35535

d = 5800830252S6003776393609366128967791759466906208965096218042
611138059385252S2358731706286910030021710859044338402170729869
760061153062025249598844480475682409662470814858171304632406 4
777048331340108509473852956450719367740611973265574242372176 1
74620776371642 07600337085333288532144708859551366702948 31

Алиса хочет передать сообщение "THIS IS TEST", которое может быть представлено числовыми значениями, используя схему кодирования 00-26 (26 — пробел).

P = 19070818260818260026190418 19

Шифрованный текст, вычисленный Алисой, — это $C = P^e$, числовое значение приведено ниже.

C = 475309123646226827206365550610545180942371796070491716523
430544529606131993285666178434183591141511974112520056829797 9
717360361012782188478927415660904800235071907152771859149751 8
658886321011483541033616578984679683867637337657774656250792 8
211481418440481418443081277305900469287424855916646210865 6
Боб может восстановить из зашифрованного текста исходный

Боб может восстановить из зашифрованного текста исходный текст, используя $P = C^d$.

P = 19070818260818260026190418 19

После расшифровки восстановленный исходный текст — "THIS IS TEST".

Приложения

Хотя *RSA* может использоваться, чтобы зашифровать и расшифровывать реальные сообщения, это — очень длинные сообщения для *RSA*.

Поэтому он является полезным для коротких сообщений. В частности мы увидим, что *RSA* применяется в цифровых подписях и других криптографических системах, которые нужны для шифрования маленьких сообщений без доступа к симметрическому ключу. Как мы увидим в последующих лекциях, *RSA* также используется для установления подлинности документа.

Криптосистемы

В этой лекции рассматриваются несколько асимметрично-ключевых криптографических систем: Рабина (Rabin), Эль-Гамаля (ElGamal), криптосистема на основе метода эллиптических кривых (ECC — Elliptic Curve Cryptosystem).

15.1. Криптосистема Рабина

Криптосистема Рабина (M. Rabin) является вариантом *криптосистемы* RSA. RSA базируется на возведении в степень сравнений. *Криптосистема* Рабина базируется на квадратичных сравнениях, и ее можно представить как криптографическую систему *RSA*, в которой значениям `e` и `d` присвоены значения `e = 2` и `d = 1/2`. Другими словами, шифрование — $C \equiv p^2 \pmod{n}$ и *дешифрование* - `P = C`$^{1/2}$ `(mod n)`.

Открытый ключ доступа в *криптосистеме* Рабина — `n`, секретный ключ является *кортежем* `(p, q)`. Каждый может зашифровать сообщение, используя `n`, но только Боб может расшифровать сообщение, используя `p` и `q`. *Дешифрование* сообщения неосуществимо для Евы, потому что она не знает значения `p` и `q`. Рисунок 15.1 показывает шифрование и *дешифрование*.

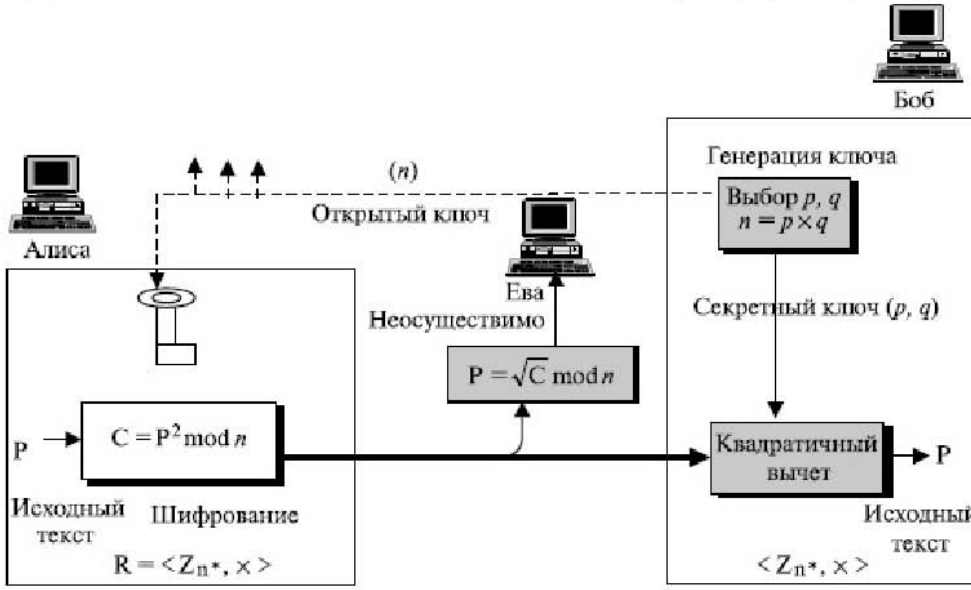


Рис. 15.1. Шифрование, дешифрование и генерация ключей в криптосистеме Рабина

Мы должны подчеркнуть, что если Боб использует *RSA*, он может сохранить `d` и `n` и отказаться после *генерации ключей* от `p`, `q` и $\varphi(n)$. Если Боб использует криптосистему Рабина, он должен сохранить `p` и `q`.

Процедура

Генерация ключей, шифрование и *дешифрование* показаны ниже.

Генерация ключей

Боб использует шаги, показанные в алгоритме 15.1, чтобы создать свой открытый ключ доступа и секретный ключ.

```
Rabin_Key_Generation
{
Выберите два больших простых числа p и q в форме 4k + 3 и p ≠ q.
 n ← p × q
Открытый_ключ ← n       // Может быть объявлен публично
Секретный_ключ ← (q, n)       // Должен сохраняться в секрете
return Открытый_ключ и Секретный_ключ
}
```

Пример 15.1. Генерации ключей для криптосистемы Рабина

Хотя два простых числа, `p` и `q`, могут быть в форме `4k + 1` или `4k + 3`, процесс *дешифрования* становится более трудным, если используется первая форма. Рекомендуют применять вторую форму, `4k + 3`, для того чтобы сделать *дешифрование* для Алисы намного проще.

Шифрование

Любой может передать сообщение Бобу, используя его открытый ключ доступа. Процесс шифрования показан алгоритмом 15.2.

```
Rabin_Encryption (n, P)       // n — открытый ключ доступа;
P — зашифрованный текст  Z*n

{
C <- P2 mod n       // C — зашифрованный текст
return C
}
```

Пример 15.2. Шифрование в криптографической системе Рабина

Хотя исходный текст `P` может быть выбран из множества Z_n, но чтобы сделать *дешифрование* более простым, мы определили множество, которое находится в Z_{n*}.

Шифрование в *криптосистеме* Рабина очень простое. Операция нуждается только в одном умножении, что может быть сделано быстро. Это выгодно, когда ресурсы ограничены: например, при использовании

карт с *интегральной схемой*, содержащей микропроцессор с ограниченной памятью, и при необходимости задействовать центральный процессор на короткое время.

Дешифрование

Боб может использовать алгоритм 15.3, чтобы расшифровать полученный зашифрованный текст.

```
Rabin_Decryption (p, q, C)      // C — зашифрованный текст; p и q — секре
a1 <- + (C^(p+1)/4) mod p
a2 <- - (C^(p+1)/4) mod p
b1 <- + (C^(q+1)/4) mod q
b2 <- - (C^(q+1)/4) mod q
// Алгоритм китайской теоремы об остатках вызывается четыре раза.
P1 <- Китайский_остаток (a1, b1, p, q)
P2 <- Китайский_остаток (a1, b2, p, q)
P3 <- Китайский_остаток (a2, b1, p, q)
P4 <- Китайский_остаток (a2, b2, p, q)
return P1, P2, P3 и P4
```

Пример 15.3. Дешифрование в криптосистеме Рабина

Мы должны подчеркнуть здесь несколько моментов. Дешифрация базируется на решении квадратичного сравнения, которое рассмотрено в лекциях 12-13. Поскольку полученный зашифрованный текст — квадрат исходного текста, это гарантирует, что C имеет корни (квадратичные *вычеты*) в Z_{n^*}. Алгоритм китайской теоремы об остатке используется, чтобы найти четыре квадратных корня.

Самый важный пункт в *криптосистеме* Рабина — это то, что она недетерминирована. *Дешифрование* имеет четыре ответа. Задача *получателя сообщения* - точно выбрать один из четырех ответов как конечный ответ. Однако во многих ситуациях получатель может легко выбрать правильный ответ.

Криптосистема Рабина не детерминирована — дешифрование создает

четыре одинаково вероятных исходных текста.

Пример 15.1

Вот очень тривиальный пример, чтобы проиллюстрировать идею.

1. Боб выбирает $p = 23$ и $q = 7$. Обратите внимание, что оба являются сравнениями $3 \bmod 4$.

2. Боб вычисляет $n = p \times q = 161$.

3. Боб объявляет n открытым и сохраняет p и q в секрете.

4. Алиса хочет передать исходный текст $P = 24$. Обратите внимание, что 161 и 24 являются взаимно простыми; 24 находится в Z_{161^*}. Она вычисляет C = от $24^2 = 93 \bmod 161$ и передает зашифрованный текст 93 Бобу.

5. Боб получает 93 и вычисляет четыре значения:

a. $a_1 = + (93^{(23+1)/4}) \bmod 23 = 1 \bmod 23$

b. $a_2 = - (93^{(23+1)/4}) \bmod 23 = 22 \bmod 23$

c. $b_1 = + (93^{(7+1)/4}) \bmod 7 = 4 \bmod 7$

d. $b_2 = - (93^{(7+1)/4}) \bmod 7 = 3 \bmod 7$

1. Боб имеет четыре возможных ответа — $(a_1, b_1), (a_1, b_2),$
2. $(a_2, b_1), (a_2, b_2)$ и использует китайскую теорему об остатках, чтобы найти четыре возможных исходных текста: $116, 24, 137$ и 45 (все из них взаимно простые к 161). Обратите внимание, что только второй ответ — исходный текст Алисы. Боб должен принять решение исходя из ситуации. Обратите внимание также, что все четыре ответа при возведении во вторую степень по модулю n дают зашифрованный текст 93, переданный Алисой.

$116^2 = 93 \bmod 161$ $24^2 = 93 \bmod 161$ $137^2 = 93 \bmod 161$ $45^2 = 93 \bmod$

Безопасность криптографической системы Рабина

Криптографическая система Рабина безопасна, пока `p` и `q` — большие числа. Сложность *криптографической системы* Рабина — такая же, как и у процедуры разложения на множители больших чисел `n` на два простых сомножителя `p` и `q`. Другими словами, *криптографическая система* Рабина так же безопасна, как и *RSA*.

15.2. Криптографическая система Эль-Гамаля

Помимо *RSA* и *криптографической системы* Рабина есть другая *криптосистема* с открытым ключом Эль-Гамаля (ElGamal), которая названа по имени ее изобретателя, Тахира Эль-Гамаля (Taher ElGamal). *Криптосистема* Эль-Гамаля базируется на свойствах *дискретного логарифма*, который обсуждался в лекциях 12-13.

Криптографическая система Эль-Гамаля

На основании сведений лекций 12-13, если `p` — очень большое простое число, e_1 — *первообразный корень* в группе $G = < Z_{p*}, \times >$ и `r` — целое число, тогда $e_2 = e_1^r \bmod p$ просто вычисляется с использованием быстрого показательного алгоритма (метод "возведения в квадрат и умножения"). Но по данным e_2, e_1 и `p`, невозможно вычислить $r = \log_{e1} e2 \bmod p$ (проблема *дискретного логарифма*).

Процедура

Рисунок 15.2 показывает *генерацию ключей*, шифрование и *дешифрование* в *криптосистеме* Эль-Гамаля.

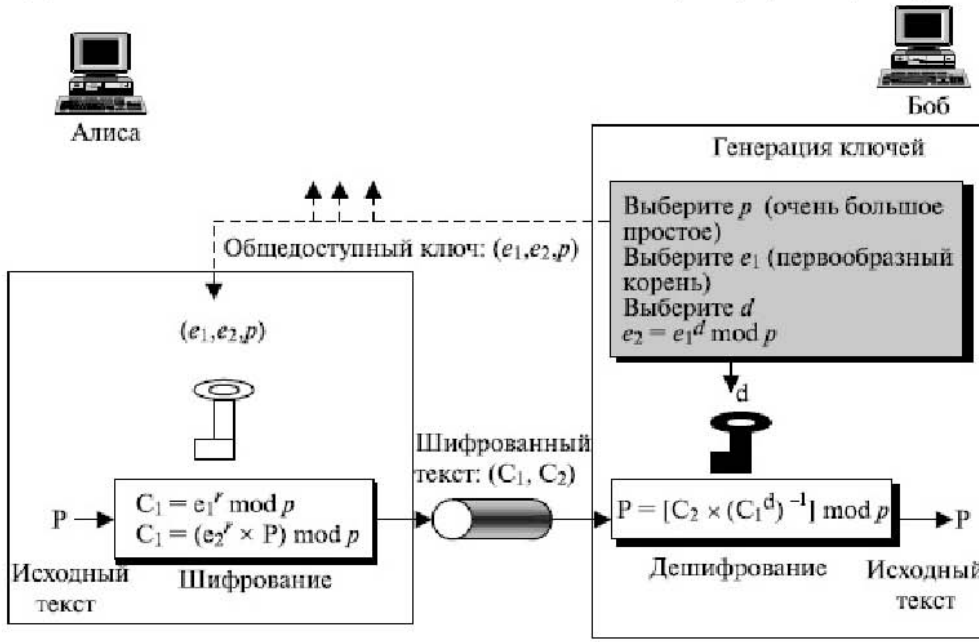


Рис. 15.2. Генерация ключей, шифрование, и дешифрование в криптосистеме Эль-Гамаля

Генерация ключей

Боб использует шаги, показанные в алгоритме 15.4, чтобы создать свои общедоступный и частный ключи.

```
ElGamal_Key_Generation
{
Выберите большое простое число p
Выберите d, члена группы G = < Zp*, x > , такое, что 1 < d < p – 2
Выберите e1 — первообразный корень в группе G = < Zp*, x >
e2 <- e1^d mod p
Общедоступный_ключ <- (e1, e2, p)     // Может быть объявлен публич
Частный_ключ <- -d                    // Должен сохраняться в секрете
return Общедоступный_ключ и Частный_ключ
}
```

Пример 15.4. Генерация ключей в криптосистеме Эль-Гамаля

Шифрование

Любой может передать сообщение Бобу, используя его открытый ключ доступа. Процесс шифрования показан в алгоритме 15.5. Если применяется быстрый показательный алгоритм (см. лекции 12-13), шифрование в *криптосистеме* Эль-Гамаля может также быть выполнено по времени с полиномиальной сложностью.

ElGamal_Encryption (e_1, e_2, p) // P — исходный текст
{
Выберите случайное целое число r в группе $G = < Z_{p*}, x >$
$C_1 <- e_1^r \bmod p$
$C_2 <- (P \times e_2^r) \bmod p$ // C_1 и C_2 – зашифрованные тексты
return C_1 и C_2 }

Пример 15.5. Шифрование в криптосистеме Эль-Гамаля

Дешифрование

Боб может использовать алгоритм 15.6, чтобы расшифровать полученное сообщение зашифрованного текста.

ElGamal_Decryption {d, p, C_1, C_2) // C_1 и C_2 — зашифрованный т
{
$P <- [C_2(C_1^d)^{-1}] \bmod p$ // P — исходный текст
return P
}

Пример 15.6. Алгоритм 15.6. Дешифрование в криптосистеме Эль-Гамаля

Сложность разрядной операции шифрования или дешифрования в криптографической системе Эль-Гамаля — полиномиальная.

Доказательство

Криптосистема Эль-Гамаля проводит дешифрацию согласно

выражению $C_2 \times (C_1{}^d)^{-1}$. Это выражение может быть проверено с помощью подстановки P:

$$\left[C_2 \times \left(C_1{}^d\right]^{-1} \bmod p = \left[(e_2{}^r \times P) \times \left(e_1{}^{rd}\right]^{-1} \bmod p = \left(e_1{}^{rd}\right) \times P \times \left(e_1{}^{rd}\right)^{-1} = P$$

Пример 15.2

Рассмотрим тривиальный пример. Боб выбирает 11 в качестве p. Затем он выбирает $e_1 = 2$. Обратите внимание, что 2 — *первообразный корень* в Z_{11^*} (см. приложение J). Затем Боб выбирает d = 3 и вычисляет $e_2 = e_1{}^d = 8$. Получены открытые ключи доступа — (2, 8, 11) и секретный ключ — 3. Алиса выбирает r = 4 и вычисляет C_1 и C_2 для исходного текста 7.

Исходный текст: 7
$C_1 = e_1{}^r \bmod 11 = 16 \bmod 11 = 5 \bmod 11$
$C_2 = (P \times e_2{}^r) \bmod 11 = (7 \times 4096) \bmod 11 = 6 \bmod 111$
Зашифрованный текст: (5, 6)

Боб получает зашифрованные тексты (5 и 6) и вычисляет исходный текст.

Зашифрованный текст: $[C_1 \times (C_2{}^d)^{-1}] \bmod 11 = 6 \times (5^3)^{-1} \bmod 11 = 6 \times 3$ m
Исходный текст: 7

Пример 15.3

Вместо того чтобы использовать $P = [C_2(C_1{}^d)^{-1}] \bmod p$ для *дешифрования*, мы можем избежать вычисления мультипликативной *инверсии* и применить $P = [C_2(C_1{}^{p-1-d})^{-1}] \bmod p$ (см. малую теорему Ферма в лекциях 12-13). В Примере 15.2 мы можем вычислить $P = [6 \times 5^{11-1-3}] \bmod 11 = 7 \bmod 11$.

Анализ

Очень интересная черта *криптосистемы* Эль-Гамаля — то, что Алиса создает r и сохраняет его в секрете; Боб создает d и сохраняет его в секрете. Это затруднение *криптографической системы* может быть решено следующим образом:

a. Алиса передает $C_2 = [e_2{}^r \times P] \mod p = [(e_1{}^{rd}) \times P] \mod p$. Выражение ($e_1{}^{rd}$) действует как маска, которая скрывает значение P. Чтобы найти значение P, Боб должен удалить эту маску.

b. Поскольку используется модульная арифметика, Боб должен создать точную копию маски и инвертировать ее (мультипликативная *инверсия*), чтобы снять воздействие маски.

c. Алиса передает Бобу $C_1 = e_1{}^r$, что является частью маски. Боб должен вычислить $C_1{}^d$, чтобы сделать точную копию маски, поскольку $C_1{}^d = (e_1{}^{r'})^d = (e_1{}^{rd})$. Другими словами, после получения точной копии маски Боб инвертирует ее и умножает результат на C_2, чтобы удалить маску.

d. Это можно представить так, что Боб помогает Алисе сделать маску $(e_1{}^{rd})$, не показывая значение d (d уже включено в $e_2 = e_1{}^{rd}$); Алиса помогает Бобу делать маску ($e_1{}^{rd}$), не раскрывая значение r (r уже включено в $C_1 = e_1{}^r$).

Безопасность криптосистемы Эль-Гамаля

Ранее были упомянуты две атаки на криптосистему Эль-Гамаля — атаки, основанные на малом значении модуля, и атаки знания исходного текста.

Атаки малого модуля

Если значение модуля p не является достаточно большим, Ева может использовать некоторые эффективные алгоритмы, чтобы решить

проблему *дискретного логарифма* и найти d или r. Если p мало, Ева может просто найти $d = \log_{e1} e_2 \bmod p$ и сохранить его, чтобы расшифровать любое сообщение, передаваемое Бобу. Это может быть сделано единожды и работать, пока Боб использует те же самые ключи. Ева может также использовать значение случайного числа r, применяемого Алисой в каждой передаче $r = \log_{e1} C_1 \bmod p$. Оба этих случая подчеркивают, что безопасность *криптосистемы* Эль-Гамаля зависит от решения проблемы *дискретного логарифма* с очень большим модулем. Поэтому рекомендовано, что p должны быть по крайней мере 1024 бита (300 десятичных цифр).

Атака знания исходного текста

Когда Алиса использует одно и то же значение случайного показателя степени r для того, чтобы зашифровать два исходных текста P и P', Ева обнаруживает P', если она знает P. Предположим, что $C_2 = P \times (e_2{}^r) \bmod p$ и $C_2' = P' \times (e_2{}^r) \bmod p$. Ева находит P', используя следующие шаги:

1. $(e_2{}^k) = C_2 \times P^{-1} \bmod p$.
2. $P' = C_2' \times (e_2{}^k)^{-1} \bmod p$.

Поэтому рекомендовано, чтобы Алиса брала при каждой передаче новое значение r, чтобы сорвать атаки.

Чтобы криптосистема Эль-Гамаля была безопасной, модуль p должен содержать по крайней мере 300 десятичных цифр, новых для каждой шифровки.

Пример 15.4

Вот более реальный пример. Боб использует случайное целое число длиной 512 битов (идеально — 1024) и целое число p длиной 155 цифр (идеал — 300 цифр). Боб выбирает e_1 и d, затем вычисляет e_2, как показано ниже; Боб объявляет (e_1, e_2, p) как свой открытый ключ и d как секретный ключ доступа.

p = 115348992725616762449253137170143317404900945326098349598119056898698622645932129754737871895144368891765264730936159292806116596434735344000857 7

---

e_1= 2

---

d = 1007

---

e_2 = 97886413043009189508766856938097739043880062887337687610032554507074156189212318317704610141673360150884132940857248531582066010072558707455

Алиса имеет исходный текст `P = 3200`, чтобы передать Бобу. Она выбирает `r = 545131`, вычисляет C_1 и C_2 и передает их Бобу.

P = 3200
r = 545131

C_1 = 88729706938352847102257047149227566312026006725656212501829417223599712681114105363661705173051581533189165400973736352957367885690606191528817

C_2 = 70845433304892994457701601238079499956743602183619244696121244696155165800779455593080345889614402408599525919579209728879681350582779566430295 0

Боб вычисляет исходный текст

$$P = C_2 \times (C_1{}^d)^{-1} \bmod p = 3200 \bmod p$$

P = 3200

Приложение

Криптосистема Эль-Гамаля может использоваться всякий раз, когда может использоваться *RSA*. Она применяется для замены ключей, установления подлинности, шифрования и *дешифрования* маленьких сообщений.

15.3. Криптосистемы на основе метода эллиптических кривых

Хотя *RSA* и Эль-Гамаль — безопасные асимметрично-ключевые *криптографические системы*, их безопасность обеспечивается ценой их больших ключей. Исследователи искали альтернативный метод, который дает тот же самый уровень безопасности с меньшими размерами ключей. Один из этих перспективных вариантов — криптосистема на основе метода эллиптических кривых (Elliptic Curve *Cryptosystem — ECC*). Система базируется на теории эллиптических кривых. Хотя глубокое рассмотрение этой теории находится вне задач и целей нашей книги, этот раздел сначала дает очень простое введение в три типа эллиптических кривых, а затем предлагает разновидности криптографических систем, которые используют некоторые из этих кривых.

Эллиптические кривые в вещественных числах

Эллиптические кривые, которые непосредственно не связаны с эллипсами, являются кубическими уравнениями двух переменных и обычно применяются для вычисления длины кривой в окружности эллипса. Общее уравнение для *эллиптической кривой*:

$$y^2 + b_1xy + b_2y = x^3 + a_1x^2 + a_2x + a_3$$

Эллиптические кривые в поле вещественных чисел используют специальный класс формы *эллиптических кривых*:

$$y^2 = x^3 + ax + b$$

В этом случае, если $4a^3 + 27b^2 \neq 0$, уравнение представляет несингулярную эллиптическую кривую ; в противоположном случае оно описывает сингулярную эллиптическую кривую. Для несингулярной *эллиптической кривой* уравнение $x^3 + ax + b = 0$ имеет три отличных корня (вещественных или комплексных); для сингулярной уравнение $x^3 + ax + b = 0$ не имеет трех отличных

корней.

В уравнении, как мы можем видеть, левая сторона (y^2) имеет степень 2, в то время как правая сторона имеет степень 3 (x^3). Это означает, что горизонтальная линия может пересекать кривую в трех точках, если все корни вещественные. Однако вертикальная линия может пересечь кривую самое большее в двух точках.

Пример 15.5

Рисунок 15.3 показывает две *эллиптические кривые* с уравнениями $y^2 = x^3 - 4x$ и $y^2 = x^3 - 1$. Оба уравнения несингулярны. Однако первое имеет три вещественных корня ($x = -2$, $x = 0$, и $x = 2$), но второе — только один вещественный корень ($x = 1$) и два мнимых.

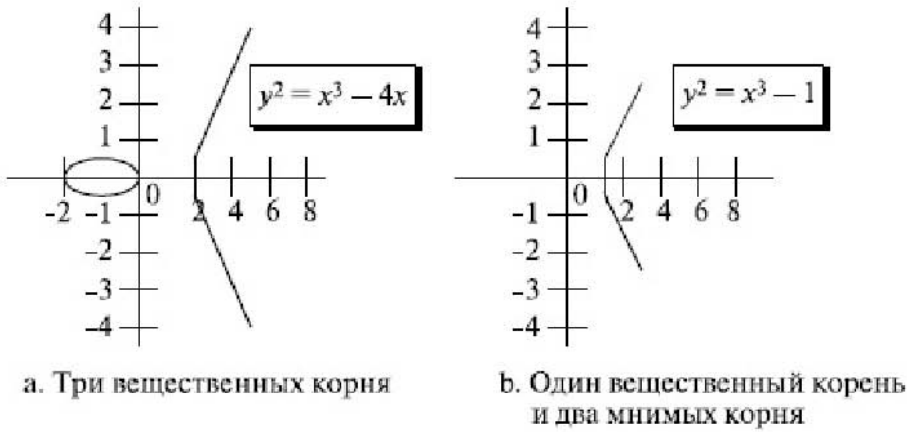


Рис. 15.3. Две эллиптические кривые в поле вещественных чисел

Абелева группа

Определим абелеву (коммутативную) группу (см. лекции 5-6), использующую точки на *эллиптической кривой*. *Кортеж* $P = (x_1, y_1)$ представляет точку на кривой, если x_1 и y_1 — координаты точки на кривой, которые удовлетворяют уравнению этой кривой. Например, точки $P = (2,0;\ 0,0)$, $Q = (0,0;\ 0,0)$, $R = (-2,0;$

0,0), S = (10,0;30,98), и T = (10,0; -30,98) – точки на кривой $y^2 = x^3 - 4x$. Обратите внимание, что каждая точка представлена двумя вещественными числами. По материалам лекций 5-6, для создания абелевой группы мы нуждаемся во множестве операций над множествами и пяти свойствах, которым удовлетворяют операции. В этом случае группа G = <E, +> — абелева.

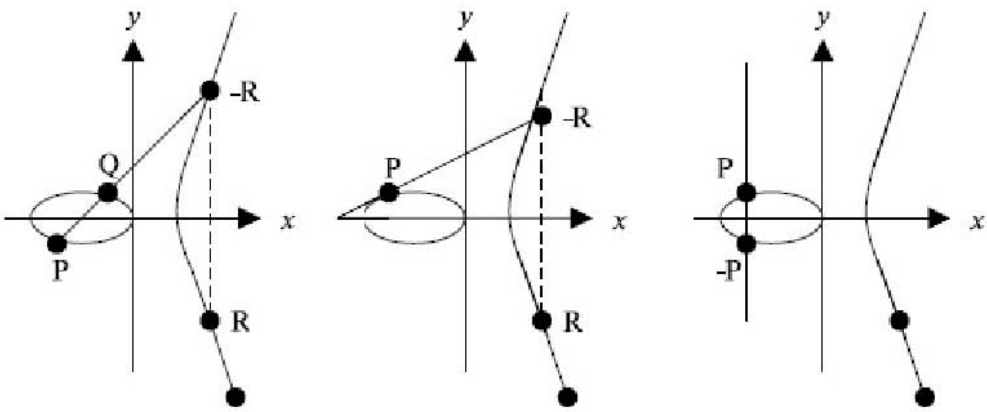
Множество. Мы определим множество как точки на кривой, где каждая точка — пара вещественных чисел. Например, множество E для *эллиптической кривой* $y^2 = -x^3 - 4x$ показано как

E = {(2,0; 0,0), (0,0; 0,0), (-2,0; 0,0), (10,0; 30,98), (10,0,-30,98)...}

Операция. Заданные свойства несингулярной *эллиптической кривой* позволяют нам определять операцию сложение точек на кривой. Однако мы должны помнить, что операция сложения здесь отличается от операции, которая была определена для целых чисел. Операция "сложение двух точек на кривой" проводится, чтобы получить другую точку на кривой.

R = P + Q, где P = (x_1, y_1), Q = (x_2, y_2), и R = (x_3, y_3)

Для того чтобы найти R на кривой, рассмотрим три случая, как это показано на рис. 15.4.



a. (R = P +Q) b. (R = P + P) c. (O = P + (-P))

Рис. 15.4. Три случая сложения на эллиптической кривой

1. В первом случае две точки $P = (x_1, y_1)$ и $Q = (x_2, y_2)$ имеют различные x -координаты и y -координаты ($x_1 \neq y_1$ и $x_2 \neq y_2$), как это показано на рис. 15.4a. Линия, соединяющая P и Q, пересекает кривую в точке, обозначенной R. R есть отражение $(-R)$ относительно y -оси. Координаты точки R, x_3 и y_3 могут быть найдены по наклону линии, λ, и затем можно вычислить значений x_3 и y_3, как показано ниже:

$$\lambda = (y_2 - y_1)/(x_2 - x_1)$$
$$x_3 = \lambda^2 - x_2 - x_1$$
$$y_3 = \lambda(x_1 - x_3) - y_1$$

2. Во втором случае две точки совпадают $(R = P + P)$, как показано на рис. 15.4b. Наклон линии и координаты точки R могут быть найдены, как показано ниже.

$$\lambda = (3x_1^2 - a)/2y_1$$
$$x_3 = \lambda^2 - x_1 - x_2$$
$$y_3 = \lambda(x_1 - x_3) - y_1$$

3. В третьем случае две точки — аддитивные *инверсии* друг друга, как это показано на рис. 15.4c. Если первая точка равна $P = (x_1, y_1)$, а вторая точка равна $Q = (x_1, -y_1)$, линия, соединяющая эти две точки, не пересекает кривую в третьей точке. Математики говорят в этом случае, что точка пересечения находится в бесконечности. Они определяют точку O (см. рис. 15.4c) как точку в бесконечности или нулевую точку, которая является аддитивным нейтральным элементом группы.

Свойства операции. Краткие определения свойств операции, как они обсуждались в лекциях 5-6:

1. Замкнутость. Может быть доказано, что сложение двух точек, с использованием операции сложения, определенное в

предыдущем разделе, создает другую точку на кривой.

2. Ассоциативность. Может быть доказано, что `(P + Q) + R = P + (Q + R)`.
3. Коммутативность. группа, состоящая из точек несингулярной *эллиптической кривой*, — абелева группа. Может быть доказано, что `P + Q = Q + P`.
4. Существование нейтрального элемента. Аддитивный нейтральный элемент в этом случае — нулевая точка. Другими словами, `P + 0 = 0 + P`.
5. Существование инверсии. Каждая точка на кривой имеет *инверсию*. *Инверсия* точки — это ее отражение относительно оси `x`. Другими словами, точки $P = (X_1, Y_1)$ И $Q = (X_1, -Y_1)$ — *инверсии* друг друга; это означает, что `P + Q = 0`. Заметим, что нейтральный элемент — это *инверсия* самого себя.

Группа и поле

Обратите внимание, что предыдущие рассуждения касаются двух алгебраических структур: группа и поле. Группа определяет множество точек на *эллиптической кривой* и операции сложения точек. Поле определяет сложение, вычитание, умножение и деление, применяющие операции над вещественными числами, которые необходимы, чтобы найти сложение точек в группе.

Эллиптические кривые в GF(p)

Наша предыдущая группа *эллиптической кривой* использовала вещественное поле для вычислений сложения точек. Криптография требует модульной арифметики. Мы определили группу *эллиптической кривой* с операцией сложения, но операция на координатах с точками в данном случае есть операция в GF(p) с `p> 3`. В модульной арифметике точки на кривой не представляют графы, как это можно было видеть на предыдущих рисунках, но сохраняются те же самые основные концепции. Мы используем ту же самую операцию сложения, но с вычислением по модулю `p`. В результате мы получаем *эллиптическую кривую* E_p `(a, b)`, где `p` определяет модуль, и `b` — коэффициент

уравнения $y^2 = x^3 + ax + b$. Обратите внимание, что хотя значение x в этом случае от 0 до p, обычно не все точки находятся на кривой.

Нахождение инверсии

Инверсия точки (x, y) равна (x, – y), где (–y) — аддитивная *инверсия* y. Например, если p = 13, *инверсия* (4, 2) равна (4, 11).

Нахождение точек на кривой

Алгоритм 15.7 показывает программу в *псевдокоде* для нахождения точек на кривой E_p (a, b).

```
Elliptic_points (p, a, b) // p-модуль
{
x ← 0
while (x < p)
{
w ← (x^3 + ax + b) mod p    //w - это y^2
if (w - целое значение квадратного корня в Z_p) выход ((x, √w)(x, –√w))
x ← x + 1
}
}
```

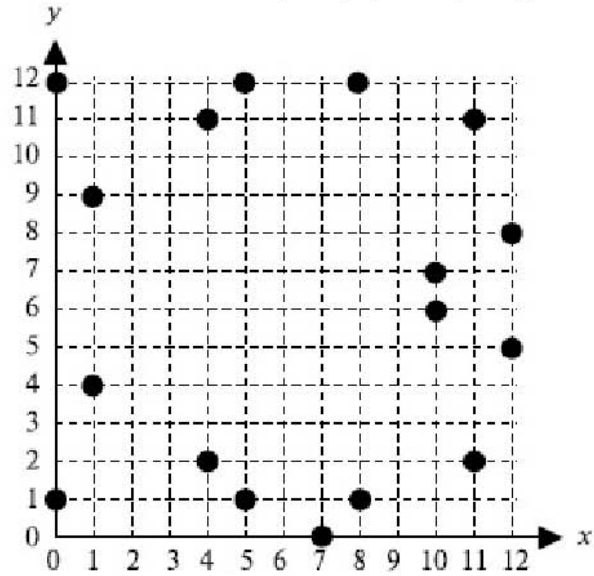
Пример 15.7. Алгоритм 15.7. Программа на псевдокоде для нахождения точек на эллиптической кривой

Пример 15.6

Определите *эллиптическую кривую* E_{13} (1, 1) по уравнению $y^2 = x^3 + x + 1$ и вычислите по модулю 13. Точки на кривой могут быть найдены, как показано на рис. 15.5.

(0,1)	(0,12)
(1,4)	(1,9)
(4,2)	(4,11)
(5,1)	(5,12)
(7,0)	(7,0)
(8,1)	(8,12)
(10,6)	(10,7)
(11,2)	(11,11)

Точки



Граф

Рис. 15.5. Точки на эллиптической кривой в поле GF (p)

Обратите внимание на следующее:

а. Некоторые значения y^2 не имеют квадратного корня по модулю 13. Они не являются точками на этой *эллиптической кривой*. Например, точки x = 2, x = 3, x = 6 и x = 9 не находятся на кривой.

б. Каждая точка, определенная на кривой, имеет *инверсию*. *Инверсии* перечислены как пары. Заметим, что (7, 0) — *инверсия* самой себя.

в. Обратите внимание, что для пары обратных точек значения y — аддитивные *инверсии* друг друга в Z_p. Например, 4 и 9 — аддитивные *инверсии* в Z_{13}. Так что мы можем сказать, что если 4 — это значение y, то 9— это значение (-y).

г. *Инверсии* находятся на тех же самых вертикальных линиях.

Сложение двух точек

Мы используем группу *эллиптической кривой*, определенную ранее, но вычисления сделаны в GF (p). Вместо вычитания и деления мы применяем аддитивные и мультипликативные *инверсии*.

Пример 15.7

Сложим две точки в примере 15.6, R = P + Q, где P = (4, 2) и Q = (10,6).

а. $X = (6 - 2) \times (10 - 4)^{-1} \bmod 13 = 4 \times 6^{-1} \bmod 13 \; != 5 \bmod 13$.

б. $x = (5^2 - 4 - 10) \bmod 13 = 11 \bmod 13$.

в. $y = [5 (4 - 11) - 2] \bmod 13 = 2 \bmod 13$.

г. R = (11, 2) является точкой на кривой в примере 15.6.

Умножение точки на константу

В арифметике умножение числа на константу k означает прибавление числа само к себе k раз. Здесь ситуация та же самая. Умножение точки P на *эллиптической кривой* на константу k означает прибавление точки P к себе k раз. Например, в E_{13} (1, 1), если точка (1, 4) умножается на 4, результат есть точка (5, 1). Если точка (8,1) умножается на 3, результат — точка (10, 7).

Эллиптические кривые в GF(2 в степени n)

Вычисление в группе *эллиптической кривой* может быть определено в поле $GF(2^n)$. В соответствии с лекциями 5-6, где мы говорили, что элементы множества в этом поле — n -битовые слова, которые можно интерпретировать как полиномы с коэффициентом в GF(2), сложение и умножение этих элементов такое же, как сложение и умножение полиномов. Для того чтобы определить *эллиптическую кривую* в $GF(2^n)$, необходимо только изменить кубическое уравнение. Общее

уравнение

$$y^2 + xy = x^3 + ax^2 + b$$

где $b \neq 0$. Обратите внимание, что значение x, y, a и b — полиномы, представляющие n -битовые слова.

Нахождение инверсии

Если P = (x, y), то (−P) = (x, x + y).

Нахождение точек на кривой

Мы можем написать алгоритм для нахождения точек на кривой, используя генераторы для полиномов, которые рассматривали в лекциях 9-10. Но разработку этого алгоритма оставляем как упражнение. Далее следует очень тривиальный пример.

Пример 15.8

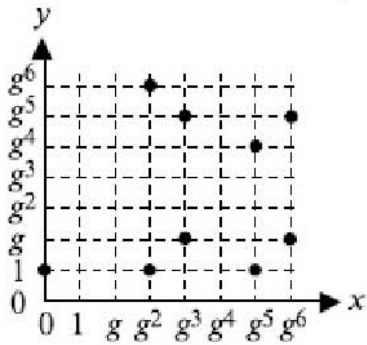
Мы выбираем GF (2 3) с элементами (0, 1, g, g^2, g^3, g^4, g^5, g^6), использующими неприводимый *полином* f (x) = x^3 + x +1. Этому соответствует *полином* g^3 + g +1 = 0 или g^3 = g + 1. Другие степени g могут быть вычислены, как это показано ниже.

0	0	$g^3 = g + 1$	0
1	0	$g^4 = g^2 + g$	1
g	0	$g^5 = g^2 + g + 1$	1
g_2	1	$g^6 = g^2 + 1$	1

Используя *эллиптическую кривую* $y^2 + xy = x^3 + g^3x^2 + 1$, a = g^3 и b = 1, мы можем найти точки на этой кривой, как это показано на рисунке 15.6.

(0,1)	(0,1)
(g^2,1)	(g^2,g^6)
(g^3,g^2)	(g^3,g^5)
(g^5,1)	(g^5,g^4)
(g^6,g)	(g^6,g^5)

Точки



Граф

Рис. 15.6. Точки на эллиптической кривой в GF (2 в степени n)

Сложение двух точек

Правила для сложения точек в GF(2^n) немного отличаются от правил GF(p).

1. Если P = (x_1, y_1), Q = (x_2, y_2), $Q \neq -P$, $Q \neq P$, то R = (x_3, y_3) = P + Q может быть найден как

$$\lambda = (y_2 + y_1)/(x_2 + x_1)$$
$$x_1 = \lambda^2 + \lambda + a$$
$$y_3 = x_1^2 + (\lambda + 1)x_3$$

2.Если Q = P, то R = P + P (или R = 2P) и может быть найден как

$$\lambda = (y_2 + y_1)/x_1$$
$$x_1 = \lambda^2 + \lambda + a$$
$$y_3 = x_1^2 + (\lambda + 1)x_3$$

Пример 15.9

Пусть нам надо найти R = P + Q, где P = (0,1) и Q = (g^2,1). Мы имеем $\lambda = 0$ и R = (g^5, g^4).

Пример 15.10

Пусть нам надо найти R = 2P, где P = $(g^2, 1)$. Мы имеем $\lambda = g^2 + 1/g^2 = g^2 + g^5 = g + 1$ и R = (g^6, g^5).

Умножение точек на константу

Для того чтобы умножить точку на константу, точки должны складываться непрерывно согласно правилу R = 2P.

Криптография эллиптической кривой, моделирующая криптосистему Эль-Гамаля

Для шифрования и *дешифрования* текстов, с помощью *эллиптических кривых* использовались несколько методов. Один из них состоит в том, чтобы моделировать криптосистему Эль-Гамаля, используя *эллиптическую кривую* в GF(p) или $GF(2^n)$, как это показано на рис. 15.7.

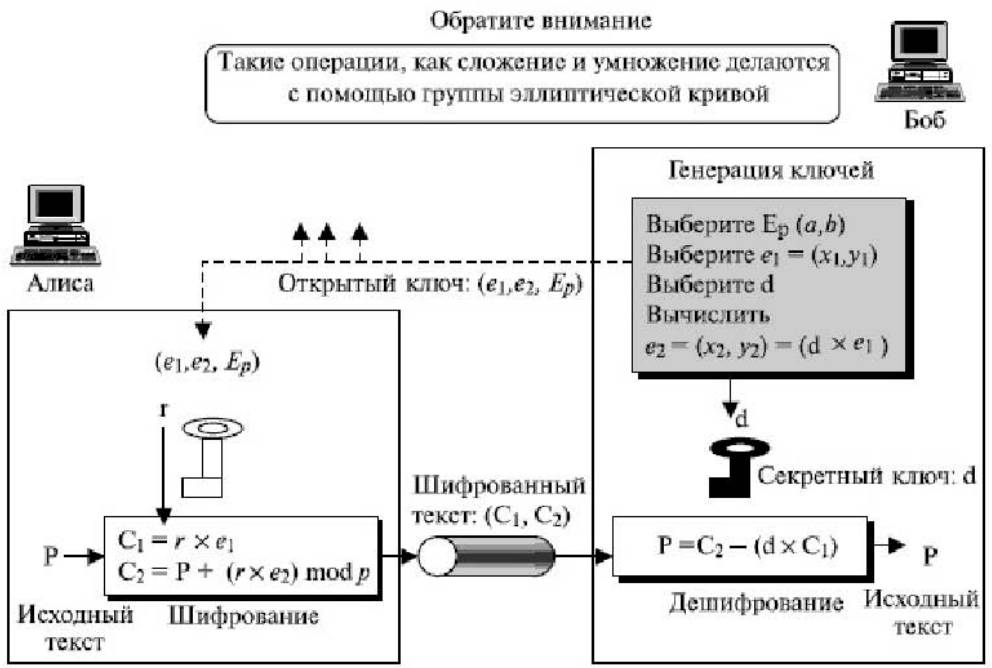


Рис. 15.7. Криптосистема Эль-Гамаля, использующая эллиптическую функцию

Генерация общедоступных и частных ключей

1. Боб выбирает E (a,b) с *эллиптической кривой* в GF(p) или $GF(2^n)$.
2. Боб выбирает точку на кривой, e_1 (x_1, y_1).
3. Боб выбирает целое число d.
4. Боб вычисляет $e_2(x_2, y_2) = d \times e_1(x_1, y_1)$. Обратите внимание: умножение здесь означает, что многократное сложение и определяется как раньше.
5. Боб объявляет E (a,b), e_1 (x_1, y_1) и e_2 (x_2, y_2) как свой открытый ключ доступа; он сохраняет d как секретный ключ.

Шифрование

Алиса выбирает P, точку на кривой, как ее исходный текст, P. Затем она вычисляет пару точек, направляет как зашифрованный текст:

Читатель может задаться вопросом, как произвольным исходным

$C_1 = r \times e_1$ $C_2 = P + r \times e_2$

Текстом может быть точка на *эллиптической кривой*. Это одна из основных проблем в применении *эллиптической кривой* для моделирования. Алиса должна использовать алгоритм, чтобы найти непосредственное соответствие между символами (или блоками текста) и точками на кривой.

Дешифрование

Боб, после получения C_1 и C_2, вычисляет P, исходный текст, используя следующую формулу:

$P = C_2 - (d \times C_1)$ Знак "минус" здесь означает сложение с инверсией.

Мы можем доказать, что P, вычисленный Бобом, — тот же, что передан Алисой, как это показано ниже:

$$P + r \times e_2 - (d \times r \times e_1) == P + (r \times d \times e_1) - (r \times d \times e_1) == P + 0 == P$$

P, C_1, C_2 и e_2 — это точки на кривой. Обратите внимание, что результат сложения двух обратных точек на кривой — нулевая точка.

Пример 15.11

Вот очень тривиальный пример шифровки с использованием *эллиптической кривой* в GF (p).

1. Боб выбирает E_{67} (2, 3) как *эллиптическую кривую* в GF (p).
2. Боб выбирает e_1 = (2, 22) и d = 4.
3. Боб вычисляет e_2 = (13, 45), где $e_2 = d \times e_1$.
4. Боб публично объявляет *кортеж* (E, e_1, e_2).
5. Алиса хочет передать исходный текст P = (24, 26) Бобу. Она выбирает r = 2.
6. Алиса находит точку C_1= (35, 1), где $C_1 = r \times e_1$.
7. Алиса находит точку C_2 = (21, 44), где $C_2 = P \times e_1$.
8. Боб получает C, и C_2. Он использует $2 \times C_1$, (35, 1) и получает (23, 25).
9. Боб инвертирует точку (23, 25) и получает точку (23, 42).
10. Боб складывает (23, 42) с C_2 = (21, 44) и получает первоначальный исходный текст P = (24, 26).

Сравнение

Ниже приводится краткое сравнение алгоритма Эль-Гамаля с его вариантом, использующим *эллиптическую кривую*.

а. Алгоритм Эль-Гамаля использует мультипликативную группу;

вариант — эллиптическую группу.

b. Эти два члена в алгоритме Эль-Гамаля — числа в мультипликативной группе; при применении варианта — точки на *эллиптической кривой*.

c. Секретный ключ в каждом алгоритме — целое число.

d. Секретные числа, выбираемые Алисой в каждом алгоритме, — целые числа.

e. Возведение в степень в алгоритме Эль-Гамаля заменено умножением точки на константу.

f. Умножение в алгоритме Эль-Гамаля заменено сложением точек.

g. *Инверсия* в алгоритме Эль-Гамаля — мультипликативная *инверсия* в мультипликативной группе; *инверсия* —заменяется аддитивной *инверсией* точки на кривой.

h. Вычисление обычно легче в *эллиптической кривой*, потому что умножение проще, чем возведение в степень, сложение проще, чем умножение, и нахождение *инверсии* намного проще в группе *эллиптической кривой*, чем в мультипликативной группе.

Безопасность метода с использованием эллептической кривой

Чтобы расшифровать сообщение, Ева должна найти значение `r` или `d`.

a. Если Ева знает значение `r`, она может использовать $P = C_2 - (r \times e_2)$, чтобы найти точку `P`, относящуюся к исходному тексту. Но для того чтобы найти `r`, Ева должна решить уравнение $C_1 = r \times e_1$. Это значит — найти две точки на кривой, C_1 и e_1. Ева должна найти множитель, который создает C_1 начиная с e_1. Эта проблема известна как проблема логарифма эллиптической кривой, единственный известный метод решения этой проблемы — $\mathrm{PO}(\rho)$ - алгоритм Поларда, который неосуществим, если задано большое `r` и `p` в

GF (p) или большое n в $GF(2^n)$.

b. Если Ева знает значение d, она может использовать $P = C_2 - (d \times C_1)$, чтобы найти точку P, относящуюся к исходному тексту. Поскольку $e_2 = d \times e_1$, это тот же самый тип проблемы, что и в предыдущем пункте. Ева знает значение e_1 и e_2 — она должна найти d.

Безопасность криптосистемы с эллиптической кривой зависит от трудности решения проблемы логарифма эллиптической кривой.

Размер модуля

Для того же самого уровня безопасности (затраты на вычисление) модуль n, может быть меньшим в эллиптической системе (*ECC*), чем в *RSA*. Например, *ECC* в $GF(2^n)$ с n, состоящим из 160 битов, может обеспечить тот же уровень безопасности, как *RSA* с n 1024 битов.

15.4. Рекомендованная литература

Для более детального изучения положений, обсужденных в этой лекции, мы рекомендуем нижеследующие книги и сайты. Пункты, указанные в скобках, показаны в списке ссылок в конце книги.

Книги

Криптографическая система RSA рассматривается в [Sti06], [Sta06], [PHS03], [Vau06], [TW06] и [Mao04]. *Криптосистемы* Рабина и Эль-Гамаля — в [Sti06] и [Mao04]. Криптография *эллиптической кривой* — в [Sti06], [Eng99] и [Bla03].

Сайты

Следующие сайты дают больше информации о темах, рассмотренных в

этой лекции.

- ссылка: http: // wwwl.ics.uci.edu / ~ mingl/knapsack.html - http://wwwl.ics.uci.edu/~mingl/knapsack.html
- ссылка: www.dtc.umn.edu/~odlyzko/doc/arch/knapsack.survey.pdf - http://www.dtc.umn.edu/~odlyzko/doc/arch/knapsack.survey.pdf
- ссылка: http://en.wikipedia.org/wiki/RSA - http://en.wikipedia.org/wiki/RSA
- ссылка: citeseer.ist.psu.edu/boneh99twenty.html - citeseer.ist.psu.edu/boneh99twenty.html
- ссылка: www.mat.uniroma3.it/users/pappa/SLIDES/RSA-HRL_05.pdf - http://www.mat.uniroma3.it/users/pappa/SLIDES/RSA-HRL_05.pdf
- ссылка: http://en.wikipedia.org/wiki/Rabin_cryptosystem - http://en.wikipedia.org/wiki/Rabin_cryptosystem
- ссылка: http://en.wikipedia.org/wiki/ElGamaL_encryption - http://en.wikipedia.org/wiki/ElGamaL_encryption
- ссылка: ww..cs.purdue.edu/homes/wspeirs/elgamal.pdf - ww..cs.purdue.edu/homes/wspeirs/elgamal.pdf
- ссылка: http://en.wikipedia.org/wiki/Elliptic__curve_cryptography - http://en.wikipedia.org/wiki/Elliptic__curve_cryptography
- ссылка: www.cs.utsa.edu/~rakbani/publications/Akbani-ECC-IEEESMC03.pdf - http://www.cs.utsa.edu/~rakbani/publications/Akbani-ECC-IEEESMC03.pdf

15.5. Итоги

- Есть два способа достигнуть информационной безопасности: криптография с *симметричными ключами* и криптография с асимметричными ключами. Эти два способа существуют параллельно и дополняют друг друга; преимущества одного могут дать компенсацию недостаткам другого.
- Концептуальные различия между этими двумя способами базируются на том, как они сохраняют секретность. В криптографии с *симметричными ключами* секретность должна быть разделена между двумя объектами; в криптографии с асимметричными ключами секретность персональная (неразделенная).
- Криптография с *симметричными ключами* базируется на

подстановке и перестановке символов; криптография с асимметричными ключами базируется на применении *математических функций* к числам.

- Криптография с асимметричными ключами использует два отдельных ключа: один секретный и один открытый. Шифрование и *дешифрование* можно представлять себе как запирание и отпирание замков ключами. Замок, который заперт открытым ключом, можно отпереть только соответствующим секретным ключом.
- В криптографии с асимметричным ключом ответственность обеспечения безопасности находится, главным образом, на плечах приемника (Боб), который должен создать два ключа: один секретный и один открытый. Боб несет ответственность за секретный ключ. Открытый ключ может быть распространен сообществу через канал распределения открытого ключа.
- В отличие от криптографии с *симметричными ключами*, в криптографии с асимметричным ключом исходный текст и зашифрованный текст обрабатываются как целые числа. Сообщение должно кодироваться как целое число (или множество целых чисел) перед шифрованием; целое число (или множество целых чисел) должно быть расшифровано в сообщение после *дешифрования*. Криптография с асимметричным ключом обычно используется, чтобы зашифровать или расшифровывать маленькие сообщения, такие как ключ шифра для криптографии с *симметричными ключами*.
- Главная идея криптографии с асимметричным ключом — понятие "лазейка" в *односторонней функции* (TOWF), которая является такой функцией, что f вычисляется просто, а f^{-1} вычислить невозможно (в смысле сложности вычислений), если не используется лазейка.
- Блестящая идея относительно криптографии общедоступного ключа принадлежит Меркелю и и Хеллману – это ранцевая *криптосистема* . Когда нам говорят, какие элементы из заранее заданного множества чисел находятся в рюкзаке, мы можем легко вычислить сумму чисел; когда нам сообщают сумму, трудно сказать, какие элементы находятся в рюкзаке, если он не заполнен элементами сверхвозрастающего множества.
- Самый общий алгоритм общедоступного ключа — *криптографическая система* RSA. *RSA* использует два числа e и

d, где e — общедоступный ключ, а d является частным (секретным). Алиса использует $C = P^e \bmod n$ для того, чтобы создать зашифрованный текст C из исходного текста P ; Боб использует $P = C^d \bmod n$, чтобы извлечь исходный текст, переданный Алисой.

- *RSA* применяет две алгебраических структуры: кольцо и группа. Шифрование и *дешифрование* выполняются с использованием *коммутативного* кольца $R =< Z_n^*, +, \times >$ с двумя арифметическими операциями — сложением и умножением. *RSA* применяет мультипликативную группу $G =< Z_n^*, \times >$ для *генерации ключей*.
- Никаких разрушительных атак на *RSA* не было обнаружено. Теоретически предсказано несколько атак, основанных на разложении на множители, выборке шифрованного текста, образце *дешифрования*, образце шифрования, исходном тексте, модуле и реализации.
- *Криптосистема* Рабина — вариант *криптографической системы* RSA. *RSA* базируется на экспоненциальном сравнении; *криптосистема* Рабина базируется на квадратичном сравнении. Мы можем представлять себе, что *криптосистема* Рабина — это *RSA*, в которой значение $e = 2$ и $d = 1/2$. *Криптографическая система* Рабина безопасна, пока p и q — большие числа. Сложность *криптосистемы* Рабина — на том же самом уровне, как и процесс разложения большого числа n на два простых сомножителя p и q.
- *Криптосистема* Эль-Гамаля базируется на проблеме *дискретного логарифма*. *Криптосистема* Эль-Гамаля использует идею *первообразных* корней в Z_n^*. Шифрование и *дешифрование* в *криптосистеме* Эль-Гамаля использует группу $G =< Z_p^*, \times >$ Общедоступный ключ — это два числа, e_1 и e_2, а секретный ключ — это целое число d. Безопасность *криптосистемы* Эль-Гамаля основана на том, что решение проблемы *дискретного логарифма* не существует. Однако в литературе была упомянута атака, основанная на малом значении модуля, и атака знания исходного текста.
- Другая *криптографическая система*, рассмотренная в этой

лекции, базируется на *эллиптических кривых*. *Эллиптические кривые* являются кубическими уравнениями в двух переменных. *Эллиптические кривые* на поле вещественных чисел используют специальный класс *эллиптических кривых* $y^2 = x^3 + ax + b$, где $4a^3 + 27b^2 \neq 0$. Абелева группа была определена с помощью *эллиптической кривой* с операцией сложения, которая показывает, как две точки на кривой можно сложить, чтобы получить другую точку на этой кривой.

- Криптография *эллиптической кривой* применяет две алгебраических структуры, абелеву группу и поле. Поле может быть полем вещественных чисел, $GF(p)$ и $GF(2^n)$. Мы показали, как *криптосистема* Эль-Гамаля может моделироваться, используя *эллиптические кривые* в конечном поле. Безопасность криптографии *эллиптической кривой* зависит от проблемы логарифма эллиптической кривой, решение которой неосуществимо при большом значении модуля.

15.6. Набор для практики

Обзорные вопросы

1. Найдите различия между криптосистемами с *симметричными ключами* и асимметричными ключами.
2. Найдите различия между открытыми и секретными ключами в *криптосистеме* с асимметричными ключами. Найдите совпадения и различие ключей в *криптосистемах* с *симметричными ключами* и с асимметричными ключами.
3. Определите "лазейку" в *односторонней функции* и объясните её использование в криптографии с асимметричным ключом.
4. Кратко объясните идею ранцевой *криптосистемы*
    - Что является односторонней функцией в этой системе?
    - Что является лазейкой в этой системе?
    - Определите открытые и секретные ключи в этой системе.
    - Опишите безопасность этой системы.
5. Кратко объясните идею *криптографической системы RSA*.
    - Что является односторонней функцией в этой системе?

- Что является лазейкой в этой системе?
- Определите открытые и секретные ключи в этой системе.
- Опишите безопасность этой системы.

6. Кратко объясните идею *криптосистемы* Рабина.
   - Что является односторонней функцией в этой системе?
   - Что является лазейкой в этой системе?
   - Определите открытые и секретные ключи в этой системе.
   - Опишите безопасность этой системы.
7. Кратко объясните идею *криптосистемы* Эль-Гамаля.
   - Что является односторонней функцией в этой системе?
   - Что является лазейкой в этой системе?
   - Определите открытые и секретные ключи в этой системе.
   - Опишите безопасность этой системы.
8. Кратко объясните идею криптографии *эллиптической кривой* (*ECC*).
   - Что является односторонней функцией в этой системе?
   - Что является лазейкой в этой системе?
   - Определите открытые и секретные ключи в этой системе.
   - Опишите безопасность этой системы.
9. Определите *эллиптические кривые* и объясните их приложения в криптографии.
10. Определите операцию, используемую в абелевой группе, которая обрабатывает точки на *эллиптической кривой*.

Упражнения

1. Учитывая сверхвозрастающий *кортеж* `b = [7, 11,23,43,87, 173, 357]`, `r =41` и модуль `n = 1001`, зашифруйте и расшифруйте букву `a`, используя ранцевую криптосистему. Используйте `[7 6 5 1 2 3 4]` как таблицу перестановки.
2. В *RSA*:
   - Дано `n = 221` и `e = 5`, найдите `d`.
   - Дано `n =3937` и `e =17`, найдите `d`.
   - Дано `p = 19`, `q = 23` и `e = 3`, найдите `n`, `φ, (n) (^)` `d`.
3. Для того чтобы понять безопасность алгоритма RSA, найдите `d`,

если вы знаете, что e =17, а n =187.

4. В *RSA* дано n и $\varphi(n)$, вычислите p и q.
5. В *RSA* дано e = 13 и n = 100. Зашифруйте сообщение "HOW ARE YOU", применяя 00 к 25 для букв от A до Z и 26 — для пробела. Используйте различные блоки, чтобы сделать P <n.
6. В *RSA* дано n = 12091 и e = 13. Зашифруйте сообщение "THIS IS THOGH", используя схему кодирования 00 к 26. Расшифровать зашифрованный текст, чтобы найти первоначальное сообщение.
7. В *RSA*:
   - Почему Боб не может выбрать 1 как открытый ключ e?
   - Какова проблема в выборе 2 открытым ключом?
8. Алиса использует открытый ключ RSA Боба (e = 17, n = 19519), чтобы передать сообщение из четырех символов Бобу, применяющему схему $A \leftrightarrow 0,\ B \leftrightarrow 1 \ldots Z \leftrightarrow 25$ кодирования и декодирования по каждому символу отдельно. Ева перехватывает зашифрованный текст (6625 0 2968 17863) и расшифровывает сообщение, не разлагая на множители модуль. Найдите исходный текст; объясните, почему Ева смогла легко взломать зашифрованный текст.
9. Алиса использует открытый ключ RSA Боба (e = 7, n = 143), чтобы передать исходный текст P = 8, зашифрованный в виде текста C = 57. Покажите, как Ева может использовать атаку выборки текста, если она имеет доступ к компьютеру Боба, чтобы найти исходный текст.
10. Алиса использует общедоступный ключ *RSA* Боба (e = 3, n = 35) и передает зашифрованный текст 22 Бобу. Покажите, как Ева может найти исходный текст, используя атаку циклического повторения.
11. Предложите, как Алиса может предотвратить атаку связанного сообщения на *RSA*.
12. Используя криптосистему Рабина с p = 47 и q =11:
    - Зашифруйте P = 17 и найдите зашифрованный текст.
    - Используя Китайскую теорему об остатках, найдите четыре возможных исходных текста.
13. В *криптосистеме* Эль-Гамаля дано простое число p = 31:
    - Выберите соответствующие e_1 и d, затем вычислите e_2.

- Зашифруйте сообщение "HELLO" ; используйте 00 к 25 для кодирования. Используйте различные блоки для того, чтобы сделать P<p.
- Расшифруйте зашифрованный текст, чтобы получить исходный текст.

14. Что случится в *криптосистеме* Эль-Гамаля —, если C_1 и C_2 будут изменены в течение передачи?
15. Предположим, что Алиса применяет в *криптосистеме* Эль-Гамаля общедоступный ключ Боба ($e_1 = 2$ и $e_2 = 8$), чтобы передать два сообщения — P = 17 и P' = 37. Они оба используют то же самое случайное целое число r = 9. Ева перехватывает зашифрованный текст и так или иначе находит значение P = 17. Покажите, как Ева может использовать атаку знания исходного текста, чтобы найти значение P'.
16. В *эллиптической кривой* E (1, 2) в поле GF (11):
    - Найдите уравнение кривой.
    - Найдите все точки на кривой и сделайте рисунок, такой же, как рис. 15.5.
    - Сгенерируйте общедоступный и секретный ключи для Боба.
    - Выберите точку на кривой как исходный текст Алисы.
    - Создайте зашифрованный текст, соответствующий исходному тексту Алисы в пункте d.
    - Расшифруйте зашифрованный текст для Боба, чтобы найти исходный текст, передаваемый Алисой.
17. В *эллиптической кривой* E (g^4, 1) в поле GF(2^4):
    - Найдите уравнение кривой.
    - Найдите все точки на кривой и сделайте рисунок, такой же, как рис. 15.5.
    - Сгенерируйте общедоступный и секретный ключи для Боба.
    - Выберите точку на кривой как исходный текст Алисы.
    - Создайте зашифрованный текст, соответствующий исходному тексту Алисы в пункте г.
    - Расшифруйте зашифрованный текст для Боба, чтобы найти исходный текст, передаемый Алисе.
18. Используйте ранцевую криптосистему:
    - Напишите алгоритм для шифрования.
    - Напишите алгоритм для *дешифрования*.
19. В *RSA*:

- Напишите алгоритм для шифрования, используя оптимальное асимметричное дополнение шифрования (OAEC).
- Напишите алгоритм для *дешифрования*, используя оптимальное асимметричное дополнение шифрования (OAEC).

20. Напишите алгоритм для атаки циклического повторения на *RSA*.
21. Напишите алгоритм для сложения двух точек на *эллиптической кривой* в $GF(p)$.
22. Напишите алгоритм для сложения двух точек на *эллиптической кривой* в $GF(2^n)$.

Список литературы

1. Barr, T, *Invitation to Cryptology*, Upper Saddle River.NJ: Prentice Hall, 2002

2. Bishop, D, *Cryptography with Java Applets*, Sudbury, MA: Jones and Bartlett, 2003

3. Bishop, M, *Computer Security*, Reading, MA: Addison-Wesley, 2005

4. Blahut, U, *Algebraic Codes for Data Transmission*, Cambridge:Cambridge University Press, 2003

5. Brassoud, D., and Wagon, S, *Computational Number Theory.Emerville*, CA: Key College, 2000

6. Coutinho, S, *The Mathematics of Ciphers*, Natick, MA: A. K.Peters, 1999

7. Dummit, D., and Foote, R, *Abstract Algebra*, Hoboken, NJ: John Wiley & Sons, 2004

8. Doraswamy, H., and Harkins, D, *PSec. Upper Saddle River*, NJ:Prentice Hall, 2003

9. Durbin, J, *Modern Algebra*, Hoboken, NJ: John Wiley & Sons,2005

10. Enge, A, *Elliptic Curves and Their Applications to Cryptography*, Norwell, MA: Kluver Academic, 1999

11. Forouzan, B, *TCP/IP Protocol Suite*, New York: McGraw-Hill,2006

12. Forouzan, B, *Data Communication and Networking*, New York:McGraw-Hill, 2007

13. Frankkel, S, *Demystifying the IPSec Puzzle*, Norwood, MA:Artech House, 2001

14. Garret, P, *Making, Breaking Codes*, Upper Saddle River, NJ:Prentice Hall, 2001

15. Kahn, D, *The Code breakers: The Story of Secret Writing*, New York: Scribner,1996

16. Kaufman, C., Perlman, R., and Speciner, M, *Network Security*, Upper Saddle River, NJ: Prentice Hall, 2001

17. Larson, R., Edwards, B., and Falvo, D, *Elementary Linear Algebra*, Boston: Houghton Mifflin, 2004

18. Mao, W, *Modern Cryptography*, Upper Saddle River, NJ: Prentice Hall, 2004

19. Menezes, A., Oorschot, P., and Vanstone, S, *Handbook of Applied Cryptography*, New York: CRC Press, 1997

20. Pieprzyk, J., Hardjono, T., and Seberry, J, *Fundamentals of Computer Security*, Berlin: Springer, 2003

21. Rescoria, E, *SSL and TLS*, Reading, MA: Addison-Wesley, 2001

22. Rhee, M, *Internet Security*, Hoboken, NJ: John Wiley & Sons,2003

23. Rosen, K, *Elementary Number Theory*, Reading, MA: Addison-Wesley, 2006

24. Solomon, D, *Data Privacy and Security*, Berlin: Springer, 2003

25. Schneier, B, *Applied Cryptography*, Reading, MA: Addison-Wesley,1996

26. Stallings, W, *Cryptography and Network Security*, Upper Saddle River, NJ: Prentice Hall, 2006

27. Stinson, D, *Cryptography: Theory and Practice*, New York: Chapman & Hall/CRC, 2006

28. Thomas, S, *SSL and TLS Essentials*, New York: John Wiley & Sons, 2000

29. Trappe, W., and Washington, L, *Introduction to Cryptography and Coding Theory*, Upper Saddle River, NJ: Prentice Hall, 2006

30. Vaudenay, S, *A Classical Introduction to Cryptography*, New York: Springer, 2006

Содержание