Введение.

В данном учебном пособии рассмотрены основные математические и криптографические понятия, необходимые для моделирования криптосистем. Учебное пособие состоит из четырех частей:

1. Основные понятия, используемые в криптографии;
2. Математические основы, используемые для анализа и построения криптосистем - теория сложности, теория информации, энтропия и неопределенность, теория чисел и другие;
3. Криптографические алгоритмы DES, AES, RSA, DSA;
4. Некоторые способы криптоанализа и ряд вопросов, связанных с анализом ключей.

Раздел 1. Основные понятия

1.1 Терминология

Отправитель и получатель

Предположим, что отправитель хочет послать сообщение получателю. Более того, этот отправитель хочет послать свое сообщение безопасно: он хочет быть уверен, что перехвативший это сообщение не сможет его прочесть. Стороны, обменивающиеся шифрованной информацией, обычно обозначаются *A* и *B*. Довольно часто употребляют более дружественные имена: Алиса и Боб. Но не следует думать, что стороны, участвующие в процессе, обязательно люди. Используя эти имена, мы вполне можем описывать обмен секретной информацией между двумя автономными механизмами. Подслушивающей стороне, «плохой девочке», которая взламывает шифротекст, обычно дают имя Ева.

Сообщения и шифрование

Само сообщение называется **открытым текстом.** Алгоритм **шифрования (или шифр)** — это перевод открытого текста в текст зашифрованный (или шифротекст, шифрограмму, криптограмму) с помощью секретного ключа. Этот процесс называют шифрованием. Обозначим открытый текст как *M* (от *message,* сообщение), или *P* (от *plaintext,* открытый текст). Это может быть поток битов, текстовый файл, битовое изображение, оцифрованный звук, цифровое видеоизображение. Для компьютера *M* - это просто двоичные данные. Открытый текст может быть создан для хранения или передачи. В любом случае, *M* - это сообщение, которое должно быть зашифровано.

Обозначим шифротекст как *C* (от *ciphertext).* Это тоже двоичные данные, иногда того же размера, что и *M,* иногда больше. Если шифрование сопровождается сжатием, *C* может быть меньше чем *M.* Само шифрование, однако, не обеспечивает сжатие информации. Функция шифрования *E* действует на *M,* создавая *C.*
Мы будем писать

$$C = E_к(M),$$

где *M* — открытый текст, *E* — шифрующая функция, *K* — секретный ключ, C — шифротекст.
Обратный процесс называют **расшифрованием** и пишут

$$M = D_k(C).$$

Заметим, что алгоритмы шифрования и расшифрования *E* и *D* открыты, и секретность исходного текста *M* в данном шифротексте *C* зависит от секретности ключа *K.*

Поскольку смыслом шифрования и последующего дешифрирования сообщения является восстановление первоначального открытого текста, должно выполняться следующее равенство:

$$D_k(E_k(M)) = M$$

Искусство и наука безопасных сообщений, называемая **криптографией,** воплощается в жизнь **криптографами. Криптоаналитиками** называются те, кто используют **криптоанализ,** искусство и науку взламывать шифротекст - раскрывать то, что было зашифровано. Отрасль математики, охватывающая криптографию и криптоанализ, называется **криптологией**, а люди, которые ей занимаются - **криптологами.** Для создания криптосистемы, криптограф должен быть хорошим криптоаналитиком.

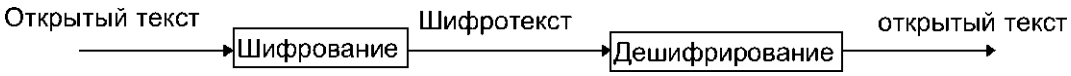


Рис. 1-1. Шифрование и дешифрирование

Проверка подлинности, целостность и неотрицание авторства

Кроме обеспечения конфиденциальности криптография часто используется для других функций:

— **Проверка подлинности.** Получатель сообщения может проверить его источник, злоумышленник не сможет замаскироваться под кого-либо.

— **Целостность.** Получатель сообщения может проверить, не было ли сообщение изменено в процессе доставки, злоумышленник не сможет подменить правильное сообщение ложным.

— **Неотрицание авторства.** Отправитель не сможет ложно отрицать отправку сообщения.

Существуют жизненно важные требования к общению при помощи компьютеров, также как существуют аналогичные требования при общении лицом к лицу. Это требует обеспечение проверки подлинности, целостности и неотрицания авторства.

Алгоритмы и ключи

Криптографический алгоритм, также называемый **шифром,** представляет собой математическую функцию, используемую для шифрования и дешифрирования. Обычно это две связанных функции: одна для шифрования, а другая для дешифрирования.

Если безопасность алгоритма основана на сохранении самого алгоритма в тайне - это **ограниченный** алгоритм. Ограниченные алгоритмы представляют только исторический интерес - они совершенно не соответствуют сегодняшним стандартам. Большая или изменяющаяся группа пользователей не может использовать такие алгоритмы, так как всякий раз, когда пользователь покидает группу, ее члены должны переходить на другой алгоритм. Алгоритм должен быть заменен и в случае, если кто-нибудь извне случайно узнает секрет.

Ограниченные алгоритмы не допускают качественного контроля или стандартизации. У каждой группы пользователей должен быть свой уникальный алгоритм. Такие группы не могут использовать открытые аппаратные или программные продукты - злоумышленник может купить такой же продукт и раскрыть алгоритм. Им приходится разрабатывать и реализовывать собственные алгоритмы.

Ограниченные алгоритмы, несмотря на эти основные недостатки, необычайно популярны для приложений с низким уровнем безопасности. Пользователи либо не понимают проблем, связанных с безопасностью своих систем, либо не заботятся о них.

Современная криптография решает эти проблемы с помощью **ключа** *K*. Такой ключ может быть любым значением, выбранным из большого множества. Множество возможных ключей называют **пространством ключей.** Шифрование и дешифрирование использует этот ключ, то есть, они зависят от ключа, что обозначается индексом *K*:

$$E_K(M)=C$$
$$D_K(C)=M$$

При этом выполняется следующее равенство:

$$D_K(E_K(M))=M$$

Для некоторых алгоритмов при шифровании и дешифрировании используются различные ключи. То есть ключ шифрования, K_1, отличается от соответствующего ключа дешифрирования, K_2. В этом случае:

$$E_{K1}(M)=C$$
$$D_{K2}(C)=M$$
$$D_{K2}(E_{K1}(M))=M$$

Безопасность этих алгоритмов полностью основана на ключах, а не на деталях алгоритмов. Это значит, что алгоритм может быть опубликован и проанализирован. Продукты, использующие этот алгоритм, могут широко тиражироваться. Не имеет значения, что злоумышленнику известен алгоритм, если ему не известен конкретный ключ, то он не сможет прочесть сообщения.

Криптосистема представляет собой алгоритм плюс все возможные открытые тексты, шифротексты и ключи.

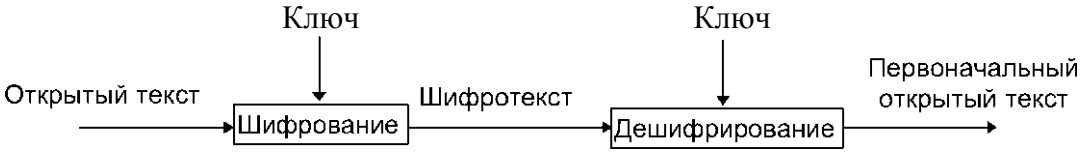


Рис. 1-2. Шифрование и дешифрирование с ключом

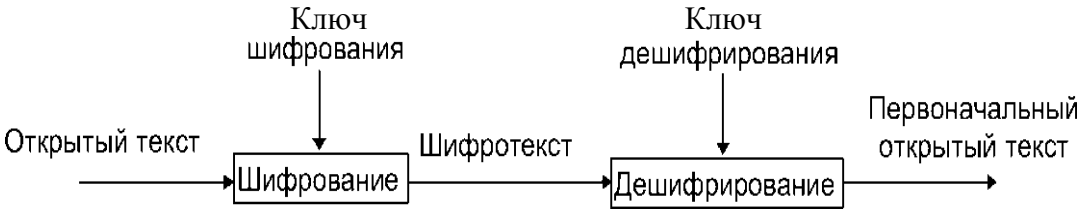


Рис. 1-3. Шифрование и дешифрирование с двумя различными ключами

1.2 Симметричные алгоритмы

Существует два основных типа алгоритмов, основанных на ключах: симметричные и с открытым ключом. **Симметричные алгоритмы,** иногда называемые условными алгоритмами, представляют собой алгоритмы, в которых ключ шифрования может быть рассчитан по ключу дешифрирования и наоборот. В большинстве симметричных алгоритмов кличи шифрования и дешифрирования одни и те же. Эти алгоритмы, также называемые алгоритмами с секретным ключом или алгоритмами с одним ключом, требуют, чтобы отправитель и получатель согласовали используемый ключ перед началом безопасной передачи сообщений. Безопасность симметричного алгоритма определяется ключом, раскрытие ключа означает, что кто угодно сможет шифровать и дешифрировать сообщения. Пока передаваемые сообщения должны быть тайными, ключ должен храниться в секрете. Шифрование и дешифрирование с использованием симметричного алгоритма обозначается как:

$$E_K(M)=C$$

$$D_K(C)=M$$

Симметричные алгоритмы делятся на две категории. Одни алгоритмы обрабатывают открытый текст побитно (иногда побайтно), они называются **потоковыми алгоритмами** или **потоковыми шифрами.** Другие работаю с группами битов открытого текста. Группы битов называются блоками, а алгоритмы - **блочными алгоритмами** или **блочными шифрами.** До появления компьютеров алгоритмы обычно обрабатывали открытый текст посимвольно. Такой вариант может рассматриваться как потоковый алгоритм, обрабатывающий поток символов. Для алгоритмов, используемых в компьютерных модемах, типичный размер блока составляет 64 бита - достаточно

большое значение, чтобы помешать анализу, и достаточно небольшое и удобное для работы.

1.3 Алгоритмы с открытым ключом

Алгоритмы с открытым ключом, называемые асимметричными алгоритмами, разработаны таким образом, что ключ, используемый для шифрования, отличается от ключа дешифрирования. Более того, ключ дешифрирования не может быть рассчитан по ключу шифрования. Алгоритмы называются "с открытым ключом", потому что ключ шифрования может быть открытым: кто угодно может использовать ключ шифрования для шифрования сообщения, но только конкретный человек с соответствующим ключом дешифрирования может расшифровать сообщение.

В этих системах ключ шифрования часто называется **открытым** ключом, а ключ дешифрирования - **закрытым.**

Открытый ключ может быть опубликован в справочнике наряду с именем пользователя. В результате любой желающий может зашифровать с его помощью свое письмо и послать закрытую информацию владельцу соответствующего секретного ключа. Расшифровать посланное сообщение сможет только тот, у кого есть секретный ключ. Более точно, имеют место преобразования:

сообщение + ОТКРЫТЫЙ КЛЮЧ АЛИСЫ = ШИФРОТЕКСТ

ШИФРОТЕКСТ + секретный ключ Алисы = сообщение.

Таким образом, каждый может послать Алисе секретную информацию, воспользовавшись ее открытым ключом. Но только Алиса в состоянии расшифровать сообщение, поскольку лишь у нее есть соответствующий секретный ключ.

Шифрование с открытым ключом *K* обозначается как:

$$E_K(M)=C$$

Хотя открытый и закрытый ключи различны, дешифрирование с соответствующим закрытым ключом об означается как:

$$D_K(C)=M$$

Иногда сообщения шифруются закрытым ключом, а дешифрируются открытым, что используется для цифровой подписи.

1.4 Криптоанализ

Смысл криптографии - в сохранении открытого текста (или ключа, или и того, и другого) в тайне от злоумышленников (также называемых взломщиками, соперниками, врагами, перехватчиками). Предполагается, что злоумышленники полностью контролируют линии связи между отправителем и получателем.

Криптоанализ - наука получения открытого текста, не имея ключа. Успешно проведенный криптоанализ может раскрыть открытый текст или ключ, он также может обнаружить слабые места в криптосистемах, что в конечном итоге приведет к предыдущему результату. Раскрытие ключа не криптологическими способами называется **компрометацией ключа.**

Основное предположение криптоанализа, впервые сформулированное в девятнадцатом веке Датчманом А. Керкхоффом (Dutchman A. Kerckhoffs) состоит в том, что безопасность полностью определяется ключом. Шифрующая, так и расшифровывающая функции общеизвестны, и тайна сообщения, при известном шифротексте, зависит только от секретности ключа.

Существует семь основных типов криптоаналитического вскрытия. Для каждого из них, предполагается, что криптоаналитик обладает всей полнотой знания о используемом алгоритме шифрования:

1. **Вскрытие с использованием только шифротекста.** У криптоаналитика есть шифротексты нескольких сообщений, зашифрованных одним и тем же алгоритмом шифрования. Задача криптоаналитика состоит в раскрытии открытого текста как можно большего числа сообщений или, что лучше, получении ключа (ключей), использованного для шифрования сообщений, для дешифрирования других сообщений, зашифрованных теми же ключами.

   Дано: $C_1=E_k(P_1)$, $C_2=E_k(P_2)$, . . . $C_i,=E_k(P_i)$.

   Получить: Либо P_1, P_2, . . . Pi; k; либо алгоритм, как получать P_i+_1 из $C_i+_1=E_k(P_i+_1)$.

2. **Вскрытие с использованием открытого текста.** У криптоаналитика есть доступ не только к шифротекстам нескольких сообщений, но и к открытому тексту этих сообщений. Его задача состоит в получении ключа (или ключей), использованного для шифрования сообщений, для дешифрирования других сообщений, зашифрованных тем же ключом (ключами).

   Дано: P_1, $C_1=E_k(P_1)$, P_2, $C_2=E_k(P_2)$, . . ., P_i, $C_i,=E_k(P_i)$.

   Получить: Либо k; либо алгоритм, как получать P_i+_1 из $C_i+_1=E_k(P_i+_1)$

3. **Вскрытие с использованием выбранного открытого текста.** У криптоаналитика не только есть доступ к шифротекстам и открытым текстам нескольких сообщений, но и возможность выбирать открытый текст для шифрования. Это предоставляет больше вариантов, чем вскрытие с использованием открытого текста, так как криптоаналитик может выбирать шифруемые блоки открытого текста, что может дать больше информации о ключе. Его задача состоит в получении ключа (или ключей), использованного для шифрования сообщений, или алгоритма, позволяющего дешифрировать новые сообщения, зашифрованные тем же ключом (или ключами).

   Дано: P_1, $C_1=E_k(P_1)$, P_2, $C_2=E_k(P_2)$, . . ., P_i, $C_i,=E_k(P_i)$, где криптоаналитик может выбирать P_1, P_2, . . . P_i

   Получить: Либо k; либо алгоритм, как получать P_i+_1 из $C_i+_1=E_k(P_i+_1)$

4. **Адаптивное вскрытие с использованием открытого текста.** Это частный случай вскрытия с использованием выбранного открытого текста. Криптоаналитик не только может выбирать шифруемый текст, но также может строить свой последующий выбор на базе полученных результатов шифрования. При вскрытии с использованием выбранного открытого текста криптоаналитик мог выбрать для шифрования только один большой блок открытого текста, при адаптивном вскрытии с использованием выбранного открытого текста он может выбрать меньший блок открытого текста, затем выбрать следующий блок, используя результаты первого выбора и так далее.

5. **Вскрытие с использованием выбранного шифротекста.** Криптоаналитик может выбрать различные шифротексты для дешифрирования и имеет доступ к дешифрированным открытым текстам. Например, у криптоаналитика есть доступ к "черному ящику", который выполняет автоматическое дешифрирование. Его задача состоит в получении ключа.

   Дано: C_1, $P_1=D_k(C_1)$, C_2, $P_2=D_k(C_2)$, . . . ,C_i, $P_i=D_k(C_i)$.

Получить: *k*

Такой тип вскрытия обычно применим к алгоритмам с открытым ключом. Вскрытие с использование выбранного шифротекста иногда также эффективно против симметричных алгоритмов. Иногда вскрытие с использованием выбранного открытого текста и вскрытие с использованием выбранного шифротекста вместе называют вскрытием с использованием выбранного текста.

6. **Вскрытие с использованием выбранного ключа.** Такой тип вскрытия означает не то, что криптоаналитик может выбирать ключ, а то, что у него есть некоторая информация о связи между различными ключами.
7. **Бандитский криптоанализ.** Криптоаналитик угрожает, шантажирует или пытает кого-нибудь, пока не получит ключ. Взяточничество иногда называется **вскрытием с покупкой ключа.** Это очень мощные способы вскрытия, часто являющиеся наилучшим путем взломать алгоритм.

Вскрытия с известным открытым текстом и с использованием выбранного открытого текста встречаются чаще, чем можно подумать. Многие сообщения имеют стандартные начало и окончание, что может быть известно криптоаналитику. Особенно уязвим шифрованный исходный код из-за частого использования ключевых слов: #define, struct, else, return. Те же проблемы и у шифрованного исполнимого кода: функции, циклические структуры и так далее. Вскрытия с известным открытым текстом (и вскрытия с выбранным шифротекстом) успешно использовались в борьбе с немцами и японцами в ходе Второй мировой войны.

Лучшими являются алгоритмы, разработанные открыто.

У криптоаналитиков не всегда есть доступ к алгоритмам, но часто они его получают. Если алгоритм используется в коммерческой программе безопасности, то это просто вопрос времени и денег, удастся ли дезассемблировать программу и раскрыть алгоритм. Если же алгоритм используется в системах военной связи, то это просто вопрос времени и денег купить (или украсть) аппаратуру и реконструировать алгоритм.

1.5 Безопасность алгоритмов.

Различные алгоритмы предоставляют различные степени безопасности в зависимости от того, насколько трудно взломать алгоритм. Если стоимость взлома алгоритма выше, чем стоимость зашифрованных данных, данный алгоритм можно считать безопасным. Если время взлома алгоритма больше, чем время, в течение которого зашифрованные данные должны сохраняться в секрете, то данный алгоритм тоже можно считать вычислительно безопасным. Если объем данных, зашифрованных одним ключом, меньше, чем объем данных, необходимый для взлома алгоритма, то тогда алгоритм, скорее всего, тоже является безопасным.

Здесь употребляется "скорее всего", потому что существует вероятность новых прорывов в криптоанализе и значимость большинства данных падает со временем. Значимость данных всегда оставалась меньше, чем стоимость взлома системы безопасности, защищающей данные.

Ларс Кнудсен (Lars Knudsen) разбил вскрытие алгоритмов по следующим категориям, приведенным в порядке убывания значимости:

1. **Полное вскрытие.** Криптоаналитик получил ключ *K*, такой, что $D_K(C) = P$.
2. **Глобальная дедукция.** Криптоаналитик получил альтернативный алгоритм, A, эквивалентный $D_K(C)$ без знания *K*.

3. **Местная (или локальная) дедукция.** Криптоаналитик получил открытый текст для перехваченного шифротекста.
4. **Информационная дедукция.** Криптоаналитик получил некоторую информацию о ключе или открытом тексте. Такой информацией могут быть несколько бит ключа, сведения о форме открытого текста и так далее.

Алгоритм является **безусловно безопасным** если, независимо от объема шифротекстов у криптоаналитика, информации для получения открытого текста недостаточно. Только шифрование одноразовым блокнотом невозможно вскрыть при бесконечных ресурсах. Все остальные криптосистемы подвержены вскрытию с использованием только шифротекста простым перебором возможных ключей и проверкой осмысленности полученного открытого текста. Это называется вскрытием **грубой силой.**

Криптография больше интересуется криптосистемами, которые тяжело взломать вычислительным способом. Алгоритм считается **вычислительно безопасным** (или, как иногда называют, сильным), если он не может быть взломан с использованием доступных ресурсов сейчас или в будущем. Термин "доступные ресурсы" является достаточно расплывчатым. Сложность вскрытия можно измерить различными способами:

1. **Сложность данных.** Объем данных, используемых на входе операции вскрытия.
2. **Сложность обработки.** Время, нужное для проведения вскрытия. Часто называется **коэффициентом работы.**
3. **Требования к памяти.** Объем памяти компьютера, необходимый для вскрытия.

В качестве эмпирического метода сложность вскрытия определяется по максимальному из этих трех коэффициентов. Ряд операций вскрытия предполагают взаимосвязь коэффициентов - более быстрое вскрытие возможно за счет увеличения требований к памяти.

Сложность выражается порядком величин. Если сложность обработки для данного алгоритма составляет 2^{128}, то 2^{128} операций требуется для вскрытия алгоритма. Если предполагается, что ваши вычислительные мощности способны выполнять миллион операций в секунду, и вы используете для решения задачи миллион параллельных процессоров, получение ключа займет свыше 10^{19} лет, что в миллиард раз превышает время существования вселенной.

В то время как сложность вскрытия остается постоянной (пока не будет придуман лучшего способа вскрытия), мощность компьютеров растет. За последние полвека вычислительные мощности феноменально выросли, и нет никаких причин подозревать, что эта тенденция не будет продолжена. Многие криптографические взломы пригодны для параллельных компьютеров: задача разбивается на миллиарды маленьких кусочков, решение которых не требует межпроцессорного взаимодействия. Объявление алгоритма безопасным, потому, что его нелегко взломать, используя современную технику, в лучшем случае ненадежно. Хорошие криптосистемы проектируются устойчивыми к взлому с учетом развития вычислительных средств на много лет вперед.

Раздел 2. Математические основы

2.1 Теория информации

Теория информации — один из краеугольных камней вычислительной техники. В данной главе изучается ее отношение к криптографии.

На первом этапе нам потребуется общее представление о разнице между теоретико-информационной стойкостью (или теоретической стойкостью) и вычислительной защищенностью (или практической стойкостью). Говоря неформально, криптографическая система называется вычислительно защищенной (или вычислительно стойкой), если наилучший из возможных алгоритмов, взламывающих ее, требует неоправданно высоких затрат вычислительных ресурсов. Принимая во внимание мощность современных компьютеров, можно считать, что 280 операций, необходимых для взлома шифра, это тот предел, выходя за который алгоритмы взлома становятся слишком дорогостоящими. Таким образом, если минимальное число N операций, необходимых алгоритму, атакующему данную криптосистему, больше 280 то говорят, что она вычислительно защищена. Заметим, что никакую реальную систему нельзя обоснованно считать вычислительно защищенной, поскольку мы не сможем доказать оптимальность найденного метода взлома. Поэтому на практике мы утверждаем ее защищенность в вычислительном отношении в том случае, если лучший из известных алгоритмов для ее взлома требует недопустимо большого количества вычислений.

Другой практический подход, связанный с вычислительной защищенностью, состоит в том, чтобы свести взлом системы к решению хорошо изученных трудных проблем. Например, мы можем попытаться показать, что вскрытие конкретной криптосистемы равносильно разложению данного большого целого числа на множители. Такие системы часто называют доказуемо стойкими. Однако при этом стойкость системы обосновывается сведением к трудной задаче, что нельзя считать корректным доказательством.

По существу, вычислительно, или доказуемо, стойкая криптосистема является стойкой по отношению к противнику, чьи вычислительные ресурсы ограничены. Даже в том случае, когда противник обладает большими, но ограниченными ресурсами, он все еще не сможет взломать систему.

При изучении вычислительно защищенных схем необходимо иметь четкие представления о некоторых проблемах:

- Нужно позаботиться о длине ключа. Если размер ключа мал, то у противника вполне может хватить ресурсов для взлома криптосистемы.
- Важно следить за последними алгоритмическими достижениями и развитием компьютерной техники.
- Мы должны быть готовы к тому, что наша криптосистема в какой-то момент будет взломана либо из-за усовершенствования аппаратных средств, либо в результате крупного алгоритмического прорыва.

Большинство криптосистем, активно эксплуатирующихся в настоящее время, вычислительно защищены. Поэтому в каждой из последующих глав мы будем уделять много внимания вычислительной стойкости изучаемых криптосистем.

С другой стороны, система называется абсолютно стойкой или совершенной, если мы не ограничиваем вычислительной мощности противника. Иначе говоря, криптосистема совершенна, если ее нельзя взломать даже с помощью бесконечного числа операций. Следовательно, независимо от алгоритмических достижений и совершенства вычислительной техники, абсолютно стойкую схему взломать невозможно. В литературе можно встретить и другой термин, закрепленный за абсолютной стойкостью, а именно, теоретико-информационная стойкость.

2.2 Энтропия и неопределенность

Основной принцип теории информации состоит в таком наблюдении: информация, по существу, то же самое, что и неопределенность. На первый взгляд, такое отождествление довольно странно, но специалисты приводят в его пользу следующий аргумент: если вы сомневаетесь в значении чего-либо, то его уточнение дает вам информацию. В применении к криптографии основной принцип можно пояснить так: предположим, что вы хотите извлечь информацию из шифротекста, другими словами, хотите знать, каково его истинное значение. При этом:

- Вам неясно, что означает данная криптограмма.
- Вы могли бы что-то предположить относительно соответствующего открытого текста.
- Уровень неизвестности, содержащийся в вашем предположении, соответствует количеству информации, находящейся в шифротексте.

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

- Воскресенье
- Понедельник
- Вторник
- Среда
- Четверг
- Пятница
- Суббота
- Не используется

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

Формально, количество информации в сообщении M измеряется энтропией сообщения, обозначаемое как H(M). Энтропия сообщения, определяющего пол, составляет 1 бит, а энтропия сообщения, определяющего день недели, немного меньше, чем 3 бита. В общем случае энтропия сообщения, измеряемая в битах, равна $\log_2 n$, где n - это количество возможных значений. При этом предполагается, что все значения равновероятны.

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

2.3 Норма языка

Для данного языка норма языка равна

$$r = H(M)/N$$

где N - это длина сообщения. При больших N норма обычного английского языка принимает различные значения от 1.0 бит/буква до 1.5 бит/буква. Клод Шеннон доказал, что энтропия зависит от длины текста. Конкретно он показал, что норма для 8-буквенных блоков равна 2.3 бит/буква, но ее значение падает и находится между 1.3 и 1.5 для 16-буквенных блоков. Томас Кавер (Thomas Cover) использовал игровую методику оценки и

обнаружил, что энтропия равна 1.3 бит/символ. Абсолютная норма языка равна максимальному количеству битов, которое может быть передано каждым символом при условии, что все последовательности символов равновероятны. Если в языке L символов, то абсолютная норма равна:

$$R = \log_2 L$$

Это максимум энтропии отдельных символов.

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

$$D=R - r$$

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

У сообщения ASCII, состоящего только из английских букв, количество информации на каждый байт составляет 1.3 бита. Значит, в каждом байте содержится 6.7 бита избыточной информации, что дает общую избыточность 0.84 бита информации на бит ASCII-текста и энтропию 0.16 бита информации на бит ASCII-текста. То же сообщение, набранное кодом BAUDOT, с 5 битами на символ, имеет избыточность 0.74 бита на бит и энтропию 0.26 бита на бит. Пробелы, пунктуация, числа и форматирование изменяют эти результаты. Возникает естественный вопрос: могут ли современные криптосистемы зашифровать текст с нулевой избыточностью? Ответ — нет: даже если современный шифр и сжимает данные, для увеличения криптостойкости он добавляет некоторую избыточную информацию к открытому тексту перед шифрованием.

2.4 Безопасность криптосистемы

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

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

Криптосистема, шифротекст в которой не дает никакой информации о соответствующем открытом тексте, называется абсолютно стойкой, или совершенной.
Зафиксируем следующие обозначения:

P — множество возможных открытых текстов, т. е. пространство сообщений;

K — совокупность возможных ключей;

C — множество шифротекстов.

Каждое из этих множеств можно понимать как пространство событий, в котором вероятности обозначаются как $p(P = M)$, $p(K = K)$, $p(C=C)$.

Определение 1. Криптосистема обладает абсолютной стойкостью, если равенство

$$p(P = M \mid C = C)=p(P = M)$$

имеет место для всех открытых текстов M, P и всех криптограмм C.

Иначе говоря, наличие шифрограммы никак не сказывается на вероятности того, что открытый текст совпадает с *M*. Вновь подчеркнем: абсолютная стойкость шифра означает, что шифротекст не несет в себе сведений об открытом тексте. Второй способ определения абсолютной стойкости дает следующая лемма.

Лемма 1. Криптосистема является абсолютно стойкой, если p(*C* = *C*|*P* =*M*)= p(*C* = *C*) для всех *M* и *C*.

Лемма 2. Для абсолютно стойкой криптосистемы имеет место неравенство:

$$\#K \geq \#C \geq \#P,$$

где #*K* — число возможных ключей, #*C* — количество возможных шифрограмм и #*P* — размер пространства сообщений.

Это подводит нас к основной теореме Шеннона, которая дает критерий абсолютной стойкости шифра.

Теорема 1. (Шеннон)

Пусть набор (*P, C, K,* $e_k(.)$, $d_k(.)$) обозначает симметричную криптосистему, в которой #*P* = #*C* = #*K*. Она обладает абсолютной стойкостью тогда и только тогда, когда использование всех ключей равновероятно;

- для каждой пары *M* и *C* существует единственный ключ *K*; такой что $e_k(M) = C$.

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

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

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

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

$$H(K) = \log_2 K$$

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

2.5 Расстояние уникальности

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

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

Шеннон определил расстояние уникальности, U, называемое также точкой уникальности, как такое приближенное количество шифротекста, для которого сумма реальной информации (энтропия) в соответствующем открытом тексте плюс энтропия ключа шифрования равняется числу используемых битов шифротекста. Затем он показал, что

имеет смысл считать, что шифротексты, которые длиннее расстояния уникальности, можно расшифровать только одним осмысленным способом. Шифротексты, которые заметно короче расстояния уникальности, скорее всего, можно расшифровать несколькими способами, каждый из которых может быть правилен, и таким образом обеспечить безопасность, поставив противника перед выбором правильного открытого текста.

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

$$U = H(K)/D$$

Расстояние уникальности является не точным, а вероятностным значением. Оно позволяет оценить минимальное количество шифротекста, при вскрытии которого грубой силой имеется, вероятно, только один разумный способ дешифрирования. Обычно чем больше расстояние уникальности, тем лучше криптосистема. Для DES с 56-битовым ключом и англоязычного сообщения, записанного символами ASCII, расстояние уникальности приблизительно равно 8.2 символа ASCII или 66 бит. В 1405-й приведены расстояния уникальности для различных длин ключа.

Расстояние уникальности измеряет не количество криптотекста, нужного для криптоанализа, а количество криптотекста, необходимое для единственности результата криптоанализа. Криптосистема может быть вычислительно неуязвима, даже если теоретически ее возможно взломать, используя малое количество шифротекста. Расстояние уникальности пропорционально избыточности. Если избыточность стремится к нулю, даже тривиальный шифр может не поддаться вскрытию с использованием только шифротекста.

Табл. 2.1. Расстояния уникальности текста ASCII, зашифрованного алгоритмами с различной длиной ключа

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

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

2.6 Практическое использование теории информации

Хотя эти понятия имеют большое теоретическое значение, реальный криптоанализ использует их достаточно редко. Расстояние уникальности гарантирует ненадежность системы, если оно слишком мало, но его высокое значение не гарантирует безопасности. Несколько практических алгоритмов абсолютно не поддаются анализу, поведение параметров теории информации могло бы способствовать взлому некоторых шифрованных сообщений. Однако, подобные соображения теории информации иногда полезны, например, для определения в конкретном алгоритме рекомендуемого интервала

изменения ключей. Криптоаналитики также используют ряд текстов не базе статистики и теории информации, чтобы выбирать наиболее перспективные направления анализа. К сожалению, большинство литературы по применению теории информации в криптоанализе остается секретной, включая основополагающую работу Алана Тьюринга (Alan Turing), написанную в 1940.

2.7 Путаница и диффузия

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

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

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

Потоковые шифры используют только путаницу, хотя ряд схем с обратной связью добавляют диффузию. Блочные алгоритмы применяют и путаницу, и диффузию. Как правило, диффузию саму по себе несложно взломать (хотя шифры с двойной перестановкой оказываются устойчивее, чем другие некомпьютерные системы).

2.8 Теория сложности

Теория сложности обеспечивает методологию анализа вычислительной сложности различных криптографических методов и алгоритмов. Она сравнивает криптографические методы и алгоритмы и определяет их безопасность. Теория информации сообщает нам о том, что все криптографические алгоритмы (кроме одноразовых блокнотов) могут быть взломаны.

Сложность алгоритмов

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

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

Временная сложность измеренная таким образом не зависит от реализации. Не нужно знать ни точное время выполнения различных инструкций, ни число битов, используемых для представления различных переменных, ни даже скорость процессора. Один

компьютер может быть на 50 процентов быстрее другого, а у третьего шина данных может быть в два раза шире, но сложность алгоритма, оцененная по прядку величины, не изменится.

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

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

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

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

С ростом n временная сложность алгоритмов может стать настолько огромной, что это повлияет на практическую реализуемость алгоритма.

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

Взглянем на проблему вскрытия алгоритма шифрования грубой силой. Временная сложность такого вскрытия пропорциональна количеству возможных ключей, которое экспоненциально зависит от длины ключа. Если n - длина ключа, то сложность вскрытия грубой силой равна $0(2^n)$. Сложность вскрытия грубой силой при 56-битовом ключе составляет 2^{56}, а при 112-битовом ключе - 2^{112}. В первом случае вскрытие возможно, а во втором - нет.

Сложность проблем

Теория сложности также классифицирует и сложность самих проблем, а не только сложность конкретных алгоритмов решения проблемы. Теория рассматривает минимальное время и объем памяти, необходимые для решения самого трудного варианта проблемы на теоретическом компьютере, известном как машина Тьюринга. Машина Тьюринга представляет собой конечный автомат с бесконечной лентой памяти для чтения-записи и является реалистичной моделью вычислений.

Проблемы, которые можно решить с помощью алгоритмов с полиномиальным временем, называются решаемыми, потому что для некоторых входных данных обычно могут быть решены за определенное время. Проблемы, которые невозможно решить за полиномиальное время, называются не решаемыми, потому что вычисление их решений быстро становится невозможным. Не решаемые проблемы иногда называют трудными. Проблемы, которые могут быть решены только с помощью суперполиномиальных алгоритмов, вычислительно не решаемы, даже при относительно малых значениях n.

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

Проблемы можно разбить на классы в соответствии со сложностью их решения. Самые важные классы и их предполагаемые соотношения показаны на рисунке 2.1. К несчастью, лишь малая часть этих утверждений может быть доказана математически.

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

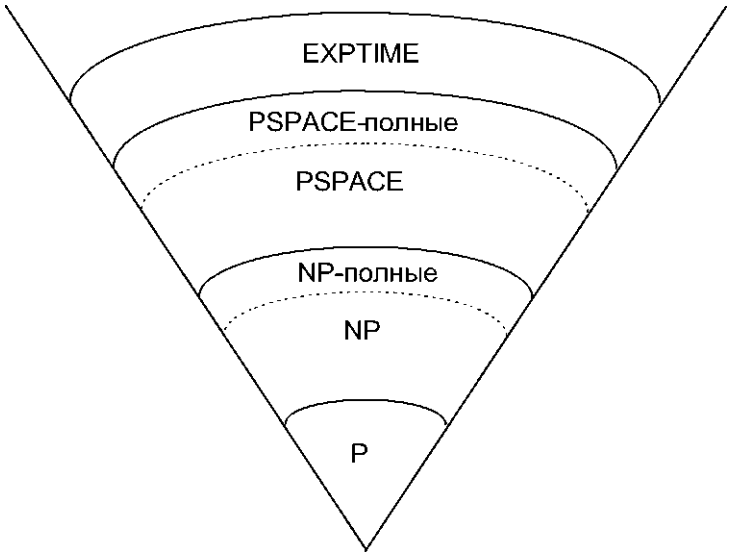


Рис. 2.1. Классы сложности

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

Класс NP включает класс P, так как любая проблема, решаемая за полиномиальное время на детерминированной машине Тьюринга, будет также решена за полиномиальное

время на недетерминированной машине Тьюринга, просто пропускается этап предположения.

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

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

С тех пор, как основополагающая работа Кука была опубликована, было показано, что существует множество проблем, эквивалентных проблеме Вопрос, верно ли P = NP, является центральным нерешенным вопросом теории вычислительной сложности, и не ожидается, что он будет решен в ближайшее время.

Следующим в иерархии сложности идет класс PSPACE. Проблемы класса PSPACE могут быть решены в полиномиальном пространстве, но не обязательно за полиномиальное время. PSPACE включает NP, но ряд проблем PSPACE кажутся сложнее, чем NP. Конечно, и это пока недоказуемо. Существует класс проблем, так называемых PSPACE-полных, обладающих следующим свойством: если любая из них является NP-проблемой, то PSPACE = NP, и если любая из них является P-проблемой, то PSPACE = P.
И наконец, существует класс проблем EXPTIME. Эти проблемы решаются за экспоненциальное время. Может быть действительно доказано, что EXPTIME-полные проблемы не могут быть решены за детерминированное полиномиальное время. Также показано, что P не равно EXPTIME.

NP-полные проблемы

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

- Проблема путешествующего коммивояжера. Путешествующему коммивояжеру нужно посетить различные города, используя только один бак с горючим (существует максимальное расстояние, которое он может проехать). Существует ли маршрут, позволяющий ему посетить каждый голод только один раз, используя этот единственный бак с горючим?
- Проблема тройного брака. В комнате n мужчин, n женщин и n чиновников. Есть список разрешенных браков, записи которого состоят из одного мужчины, одной женщины и одного регистрирующего чиновника. Дан этот список троек, возможно ли построить n браков так, чтобы любой либо сочетался браком только с одним человеком или регистрировал только один брак?
- Тройная выполнимость. Есть список n логических выражений, каждое с тремя переменными. Например: если (x и y) то z, (x и w) или (не z), если ((не и и не x) или (z и (и или не x))) то (не z и и) или x), и т.д. Существует ли правильные значения всех переменных, чтобы все утверждения были истинными? (Это частный случай упомянутой выше проблемы Выполнимости.)

2.9 Теория чисел

Арифметика вычетов

Арифметика остатков или арифметика вычетов является основой современной криптографии. Иногда ее называют "арифметикой часов". Если Милдред сказала, что она будет дома к 10:00, и опоздала на 13 часов, то когда она придет домой? Это арифметика по модулю 12. Двадцать три по модулю 12 равно 11.

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

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

10 + 13 = 11 (mod 12)

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

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

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

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

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

(a + b) mod n == ((a mod n) + (b mod n)) mod n

(a - b) mod n == ((a mod n) - (b mod n)) mod n

(a * b) mod n == ((a mod n) * (b mod n)) mod n

(a * (b+c)) mod n == (((a*b) mod n) + ((a*c) mod n)) mod n

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

a*x mod n,

представляет собой просто последовательность умножений и делений, но существуют приемы, ускоряющие это действие. Один из таких приемов стремится минимизировать количество умножений по модулю, другой - оптимизировать отдельные умножения по модулю. Так как операции дистрибутивны, быстрее выполнить возведение в степень как поток последовательных умножений, каждый раз получая вычеты.

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

(a * a * a * a * a * a * a * a) mod n
Вместо этого выполните три меньших умножения и три меньших приведения по модулю:
$((a^2 \bmod n)^2 \bmod n)^2 \bmod n$
Точно также,
$a^{16} \bmod n = (((a^2 \bmod n)^2 \bmod n)^2 \bmod n)^2 \bmod n$
Вычисление a^x, где x не является степенью 2, не на много труднее. Двоичная запись представляет x в виде суммы степеней 2: 25 - это бинарное 11001, поэтому 25 = 24 + 23 + 20.
Поэтому $a^{25} \bmod n = (a*a^{24}) \bmod n = (a* a^8*a^{16}) \bmod n =$
$= (a*((a^2)^2)^2*(((a^2)^2)^2)^2) \bmod n = (a*(((a*a^2)^2)^2)^2) \bmod n$
С продуманным сохранением промежуточных результатов вам понадобится только шесть умножений:
$(((((((a^2 \bmod n)* a)^2 \bmod n)^2 \bmod n)^2 \bmod n)^2 \bmod n)^2 *a) \bmod n$
Такой прием называется цепочкой сложений, или методом двоичных квадратов и умножения. Он использует простую и очевидную цепочку сложений, в основе которой лежит двоичное представление числа.

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

Эффективным способом много раз выполнять приведение по модулю для одного n является метод Монтгомери. Другой метод называется алгоритмом Баррета. Алгоритм Баррета является наилучшим для малых аргументов, а метод Монтгомери - наилучшим для обычного возведения в степень по модулю. Метод Монтгомери также использует преимущество малых показателей степени, используя прием, называющийся смешанной арифметикой.
Операция, обратная возведению в степень по модулю n, вычисляет дискретный логарифм.

Простые числа

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

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

Наибольший общий делитель

Два числа называются взаимно простыми, если у них нет общих множителей кроме 1. Иными словами, если наибольший общий делитель a и n равен 1. Это записывается как:
НОД(a,n)=1

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

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

Алгоритм Евклида.

Здесь мы рассмотрим только случай целых чисел. Распространить его на многочлены несложно, поскольку как целые числа, так и многочлены обладают свойством евклидовости: их можно делить с остатком. Разделить целое число a на число b с остатком — это значит найти такие числа q и r с $0 \le r < |b|$, при которых выполняется равенство

$$a = q \cdot b + r.$$

Если мы хотим разделить многочлен f на многочлен g с остатком, то нам нужно найти многочлены g и r, такие что $0 \le \deg r < \deg g$ и

$$f = q.g + r.$$

Для вычисления наибольшего общего делителя чисел $r_0 = a$ и $r_1 = b$ мы последовательно вычисляем $r_2, r_3 \ldots$, производя деление с остатком по следующей схеме:

$$\begin{aligned} r_2 &= q_1 r_1 + r_0, \\ r_3 &= q_2 r_2 + r_1, \\ &\ldots \ \ldots \\ r_m &= q_{m-1} r_{m-1} + r_{m-2}, \\ r_{m+1} &= q_m r_m. \end{aligned}$$

Очевидно, что если число d делит как a, так и b, то оно делит и все r_i, начиная с i=0 и заканчивая i=m. Следовательно,

$$НОД(a,b) = НОД(r_0,r_1) = \ldots = НОД(r_{m-1},r_m) = r_m$$

Продемонстрируем работу алгоритма на вычислении НОД (21,12), равного, без сомнения, 3. Используя описанную схему, мы находим искомый делитель за несколько шагов: НОД (21,12) = НОД (21 (mod 12), 12) = НОД (9,12) = НОД(12 (mod 9), 9) = НОД (3,9) = = НОД (9 (mod3), 3) = НОД (0,3) = 3.

Приведем пример и с большими числами.

НОД (1426 668 559 730,810 653 094 756) = НОД (810 653 094 756,616 015 464 974) = =НОД (616 015 464 974,194 637 629 782) = НОД (194 637 629 782,32102 575 628) = НОД (32 102 575 628,2 022 176 014) = НОД (2 022 176 014,1769 935 418) = НОД (1 769 935 418,252 240 596) = НОД (252 240 596,4 251246) = НОД (4 251246,1417 082) = НОД (1417082,0) = 1417082.

Работа алгоритма Евклида основана на том, что отображение

$$(a, b) \to (a \ (\text{mod } b), b)$$

сохраняет наибольший общий делитель. Неприятность состоит в том, что компьютерам намного легче складывать и умножать числа, чем вычислять остатки и частные. Поэтому реализация алгоритма с приведенным выше отображением обычно не столь эффективна, как хотелось бы. Однако найден ряд других подходящих отображений, которые также сохраняют НОД, но более экономичны с точки зрения компьютера, например,

$$(a,b) \to \begin{cases} ((a - b)/2, b), & \text{если a и b нечетные;} \\ (a/2, b), & \text{если a четное, a b нечетное;} \\ (a, b/2), & \text{если a нечетное и b четное.} \end{cases}$$

Напомним, что компьютерам делить на 2 довольно легко, поскольку в двоичной системе счисления эта операция равносильна простому сдвигу разрядов. Последнее отображение дает основу для двоичного алгоритма Евклида, который обычно и реализуется в компьютерных программах. По существу, этот алгоритм использует последнее отображение после того, как выделит из НОД максимально возможную степень двойки.

Обратные значения по модулю.

Помните, что такое обратные значения? Обратное значение для 4 это 1/4, потому что 4*1/4 =1.
В мире вычетов проблема усложняется: 4*x = 1 (mod 7)
Это уравнение эквивалентно обнаружению x и k, таких что 4x = 7k + 1, где x и k - целые числа. Общая задача состоит в нахождении x, такого что 1 = (a*x) mod n
Это также можно записать как a^{-1} = x (mod n)

Проблему обратных значений по модулю решить нелегко. Иногда у нее есть решение, иногда нет. Например, обратное значение 5 по модулю 14 равно 3. С другой стороны у числа 2 нет обратного значения по модулю 14.
В общем случае у уравнения a^{-1} = x (mod n) существует единственное решение, если a и n взаимно просты. Если a и n не являются взаимно простыми, то a^{-1} = x (mod n) не имеет решений. Если n является простым числом, то любое число от 1 до n -1 взаимно просто с n и имеет в точности одно обратное значение по модулю n.

Существует два пути. Обратное значение a по модулю n можно вычислить с помощью алгоритма Эвклида. Иногда это называется расширенным алгоритмом Эвклида.
Алгоритм итеративен и для больших чисел может работать медленно. Кнут показал, что среднее число выполняемых алгоритмом делений равно: $0.843*\log_2(n) + 1.47$

Расширенный алгоритм Евклида.

С помощью алгоритма Евклида, вычисляя НОД (a, N), мы можем выяснить, обратимо ли число a по модулю N. Однако мы до сих пор не знаем, как же найти обратный к a элемент, даже если он существует. Напомним, что алгоритм Евклида — это последовательное деление с остатком, где

$$r_{i-2} = q_{i-1}r_{i-1} + r_i, \quad i = 0, 1, \ldots, m,$$

где $r_0 = a,\ r_1 = b$ и $r_m = \text{НОД}\,(a, b)$. Сейчас мы преобразуем эти формулы, выразив все остатки r_i через a и b.

$$r_2 = r_0 - q_1 r_1 = a - q_1 b,$$
$$r_3 = r_1 - q_2 r_2 = b - q_2(a - q_1 b) = -q_2 a + (1 + q_1 q_2)b,$$
$$\ldots\ \ldots$$
$$r_{i-2} = s_{i-2}a + t_{i-2}b,$$
$$r_{i-1} = s_{i-1}a + t_{i-1}b,$$
$$r_i = r_{i-2} - q_{i-1}r_{i-1} =$$
$$= a(s_{i-2} - q_{i-1}s_{i-1}) + b(t_{i-2} - q_{i-1}t_{i-1}),$$
$$\ldots\ \ldots$$
$$r_m = s_m a + t_m b.$$

Расширенный алгоритм Евклида по данным целым числам a и b выдает r_m, s_m, и t_m, так что

$$r_m = \text{НОД}\,(a, b) = s_m a + t_m b.$$

Теперь мы готовы решить исходную задачу по определению обратного элемента для a по модулю N, если это в принципе возможно сделать. Сначала мы применяем расширенный алгоритм Евклида к числам a и N и получаем такие числа d, s и t, что
d = НОД (a, N) =s·a + t·N.
Значит, d = sa +tN = sa (mod N). Теперь видно, что уравнение ax = 1 (mod N) имеет решение только тогда, когда d = 1. При этом решение имеет вид x = a^{-1}= s.
В качестве примера вычислим обратный элемент к 7 по модулю 19. Положим r_1 = 7, r_0= 19 и проведем описанную процедуру.

$r_2 = 5 = 19 - 2 \cdot 7$,
$r_3 = 2 = 7 - 5 = 7-(19- 2-7) = -19 + 3-7$
$r_4 = 1 = 5- 2- 2 = (19- 2-7)- 2- (-19 + 3 \cdot 7) = 3 \cdot 19 - 8 \cdot 7$.
Отсюда

$$1 = -8 \cdot 7 \pmod{19},$$

так что

$$7^{-1} = -8 = 11 \pmod{19}.$$

Малая теорема Ферма

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

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

Пьер де Ферма (Pierre de Fermat), французский математик, жил с 1601 по 1665 год.

Функция Эйлера

Существует другой способ вычислить обратное значение по модулю n, но его не всегда возможно использовать. Приведенным множеством остатков mod n называется подмножество полного множества остатков, члены которого взаимно просты с n. Например, приведенное множество остатков mod 12 - это {1, 5, 7, 11}. Если n - простое число, то приведенное множество остатков mod n - это множество всех чисел от 1 до n-1. Для любого n, не равного 1,число 0 никогда не входит в приведенное множество остатков. Функция Эйлера, которую также называют функцией фи Эйлера и записывают как Ф(n), - это количество элементов в приведенном множестве остатков по модулю n. Иными словами, Ф(n) - это количество положительных целых чисел, меньших n и взаимно простых с n (для любого n, большего 1). (Леонард Эйлер (Leonhard Euler), швейцарский математик, жил с 1707 по 1783 год.)

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

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

Теперь легко вычислить $a^{-1} \bmod n$: $x = a^{\Phi(n)-1} \bmod n$

Например, какое число является обратным для 5 по модулю 7? Так как 7 - простое число, Ф(7) = 7 - 1 = 6. Итак, число, обратное к 5 по модулю 7, равно $5^{6-1} \bmod 7 = 5^5 \bmod 7 = 3$

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

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

Используя обобщение Эйлера, решаем $x = (b* a^{\Phi(n)-1}) \bmod n$

Используя алгоритм Эвклида, находим $x = (b* (a^{-1} \bmod n)) \bmod n$

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

Китайская теорема об остатках

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

В общем случае, если разложение числа n на простые сомножители представляет собой $p_1*p_2*...*p_t$, то система уравнений $(x \bmod p_i) = a_i$, где i = 1, 2, . . . , t имеет единственное решение, x, меньшее n. Обратите внимание, что некоторые простые числа могут появляться несколько раз. Например, p_1 может быть равно p_2. Другими словами, число

(меньшее, чем произведение нескольких простых чисел) однозначно определяется своими остатками от деления на эти простые числа.

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

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

Для получения x сначала воспользуемся алгоритмом Эвклида, чтобы найти u, такое что u*q = 1 (mod p) Затем вычислим: x = (((a - b) *u) mod p) * q + b

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

Если a >b mod p, то x = (((a - (b mod p)) * u) mod p) * q + b

Если a < b mod p, то x = (((a + p - (b modp))*u) modp)*q + b

Квадратичные вычеты

Если p - простое число, и a больше 0, но меньше p, то a представляет собой квадратичный вычет по модулю p, если $x^2 = a \pmod p$, для некоторых x.

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

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

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

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

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

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

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

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

$x^2 = 3 \pmod 7$

$x^2 = 5 \pmod 7$

$x^2 = 6 \pmod 7$

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

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

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

Символ Лежандра

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

L(a,p)= 0, если a делится на p.

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

L(a,p) = -1, если a не является квадратичным вычетом по модулю p.
L(a,p) можно рассчитать следующим образом: L(a,p) = a(p-1)/2 mod p
Или можно воспользоваться следующим алгоритмом:

Если a = 1, то L(a,p) = 1

Если a четно, то L(a,p)= L(a/2,p) * (-1)(e2-1)/8

Если a нечетно (и ≠ 1), то L(a,p)= L(p mod a, p)*(-1)(a-1)(p-1)/4

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

Символ Якоби

Символ Якоби, J(a,n), представляет собой обобщение символа Лежандра на составные модули, он определяется для любого целого a и любого нечетного целого n. Функция удобна при проверке на простоту. Символ Якоби является функцией на множестве полученных вычетов делителей n и может быть вычислен по различным формулам.

Вот один из способов:

Определение 1: J(a,n) определен, только если n нечетно.
Определение 2: J(0, n) = 0.
Определение 3: Если n - простое число, то символ Якоби J(a,n) = 0, если a делится на n.
Определение 4: Если n - простое число, то символ Якоби J(a,n) = 1, если a - квадратичный вычет по модулю n.
Определение 5: Если n - простое число, то символ Якоби J(a,n) = -1, если a не является квадратичным вычетом по модулю n.
Определение 6: Если n - составное число, то символ Якоби J(a,n) = J(a,p1)* ... * J(a,pm), где p_1... , p_m - это разложение n на простые сомножители.
Следующий алгоритм рекурсивно рассчитывает символ Якоби:
Правило 1: J(1,n) = 1
Правило 2: J(a*b,n) = J(a,n)* J(b,n)
Правило 3: J(2,n) =, если (n2-1) /8 нечетно, и -1 в противном случае
Правило 4: J(a,n)= J((a mod n),n)
Правило 5: J(a, b1*b2) = J(a, b1)* J(a, b2)
Правило 6: Если наибольший общий делитель a и b = 1, а также a и b нечетны:
Правило 6a: J(a,b)= J(b, a), если (a - l)(b - 1)/4 четно
Правило 6b: J(a,b)= -J(b, a), если (a - l)(b - 1)/4 нечетно
Если заранее известно, что n - простое число, вместо использования предыдущего алгоритма просто вычислите a((n-1)/2) mod n, в этом случае J(a,n) эквивалентен символу Лежандра.

Символ Якоби нельзя использовать для определения того, является ли a квадратичным вычетом по модулю n (если, конечно, n не является простым числом). Обратите внимание, что если J(a,n) = 1 и n - составное число, то утверждение, что a является квадратичным вычетом по модулю n, не обязательно будет истиной.
Например: J(7,143) = J(7,11)* J(7,13) = (-1)(-1) = 1
Однако не существует таких целых чисел x, что x^2 =7 (mod 143).

Целые числа Блюма

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

Генераторы

Если p - простое число, и g меньше, чем p, то g называется генератором по модулю p, если для каждого числа b от 1 до p - 1 существует некоторое число a, что $g^a = b \pmod p$. Иными словами, g является примитивом по отношению к p. Например, если p = 11, то 2 - это генератор по модулю 11:

$2^{10} = 1024 = 1 \pmod{11}$
$2^1 = 2 = 2 \pmod{11}$
$2^8 = 256 = 3 \pmod{11}$
$2^2 = 4 = 4 \pmod{11}$
$2^4 = 16 = 5 \pmod{11}$
$2^9 = 512 = 6 \pmod{11}$
$2^7 = 128 = 7 \pmod{11}$
$2^3 = 8 = 8 \pmod{11}$
$2^6 = 64 = 9 \pmod{11}$
$2^5 = 32 = 10 \pmod{11}$

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

В общем случае проверить, является ли данное число генератором, нелегко. Однако задача упрощается, если известно разложение на множители для p - 1. Пусть $q_1, q_2, \ldots, q_n$ - это различные простые множители p-1. Чтобы проверить, является ли число g генератором по модулю p, вычислите $g^{(p-1)/q} \bmod p$ для всех значений $q = q_1, q_2, \ldots, q_n$
Если это число равно 1 для некоторого q, то g не является генератором. Если для всех значений q рассчитанное значение не равно 1, то g - это генератор.

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

$2^{(11-1)/5} \pmod{11} = 4$
$2^{(11-1)/2} \pmod{11} = 10$

Ни один из ответов не равен 1, поэтому 2 - это генератор.
Проверим, является генератором ли число 3:

$3^{(11-1)/5} \pmod{11} = 9$
$3^{(11-1)/2} \pmod{11} = 1$

Следовательно, 3 - это не генератор.
При необходимости обнаружить генератор по модулю p просто случайно выбирайте число от 1 до p - 1 и проверяйте, не является ли оно генератором. Генераторов достаточно, поэтому один из них вы, скорее всего, найдете быстро.

2.10 Вычисление в поле Галуа

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

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

Арифметика поля Галуа широко используется в криптографии. В нем работает вся теория чисел, поле содержит числа только конечного размера, при делении отсутствуют

ошибки округления. Многие криптосистемы основаны на GF(p), где p - это большое простое число.

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

Если вы хотите попробовать с неприводимыми многочленами, то $GF(2^3)$ включает следующие элементы: 0, 1, x, x + 1, x^2, x^2 + 1, x^2 + x, x^2 + x + 1.

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

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

Вычисления в $GF(2^n)$ могут быть быстро реализованы аппаратно с помощью сдвиговых регистров с линейной обратной связью. По этой причине вычисления над $GF(2^n)$ часто быстрее, чем вычисления над GF(p). Так как возведение в степень в $GF(2^n)$ гораздо эффективнее, то эффективнее и вычисление дискретных логарифмов.

Для поля Галуа $GF(2^n)$ криптографы любят использовать в качестве модулей трехчлены $p(x) = x^n + x + 1$, так как длинная строка нулей между коэффициентами при x^n и x позволяет просто реализовать быстрое умножение по модулю. Полином должен быть примитивным, в противном случае математика не будет работать. $x^n + x + 1$ примитивен для следующих значений n, меньших чем 1000:
1, 3, 4, 6, 9, 15, 22, 28, 30, 46, 60, 63, 127, 153, 172, 303, 471, 532, 865, 900
Существуют аппаратные реализации $GF(2^{127})$, где $p(x) = x^{127} + x + 1$.

2.11 Разложение на множители

Разложить число на множители - значит найти его простые сомножители.
10 = 2*5
60 = 2*2*3*5
252601 =41*61*101
2^{113}- 1 =3391*23279*65993*1868569*1066818132868207
Разложение на множители является одной из древнейших проблем теории чисел, также это называется задачей факторизации. Этот процесс несложен, но требует времени. Это пока остается так, но ряд сдвигов в этом искусстве все же произошел. Сегодня самым лучшим алгоритмом является:

Решето числового поля чисел (Number field sieve, NFS). Решето общего числового поля - это самый быстрый из известных алгоритм для чисел размером 110 и более разрядов. В своем первоначальном виде он был непрактичен, но за последние несколько лет он был последовательно улучшен. NFS все еще слишком нов, чтобы бить рекорды разложения на множители, но скоро все переменится. Ранняя версия использовалась для разложения на множители девятого числа Ферма: $2^{512} + 1$.
Другие алгоритмы, вытесненные NFS:

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

Метод эллиптической кривой (Elliptic curve method, ECM). Этот метод использовался для поиска не более, чем 43-разрядных множителей.

Алгоритм Монте-Карло Полларда (Pollard's Monte Carlo algorithm).

Алгоритм непрерывных дробей (Continued fraction algorithm). Этот алгоритм не подходит по времени выполнения.

Проверка делением (Trial division). Этот самый старый алгоритм разложения на множители состоит из проверки каждого простого числа, меньшего или равного квадратному корню из раскладываемого числа. В качестве хорошего введения в различные алгоритмы разложения на множители, кроме NFS.

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

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

NFS намного быстрее, оценка его эвристического времени выполнения:

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

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

В 1988 году Карл Померанс (Carl Pomerance), используя обычные СБИС, спроектировал устройство для разложения на множители. Размер числа, которое можно было разложить, зависел только от размеров устройства, которое так и не было построено.

В 1993 году с помощью квадратичного решета было разложено на множители 120-разрядное трудное число. Расчет, потребовавший 825 mips-лет, был выполнен за три месяца реального времени.

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

В марте 1994 года с помощью двойной вариации множественного полиномиального QS командой математиков под руководством Ленстры было разложено на множители 129-разрядное (428-битовое) число. Вычисления выполнялись добровольцами в Internet - в течение восьми месяцев трудились 600 человек и 1600 компьютеров, возможно, самый большой в истории многопроцессорный конгломерат. Трудоемкость вычислений была в диапазоне от 4000 до 6000 mips-лет. Компьютеры соединялись по электронной почте, передавая свои результаты в центральное хранилище, где выполнялся окончательный анализ. В этих вычислениях использовались QS и теория пятилетней давности, NFS мог бы ускорить выполнение расчетов раз в десять.

С целью развития искусства разложения на множители RSA Data Security, Inc. в марте 1991 года объявило о программе RSA Factoring Challenge (состязание RSA по разложению на множители). Состязание состоит в разложении на множители ряда трудных чисел, каждое из которых является произведением двух простых чисел примерно одинакового размера. Каждое простое число было выбрано конгруэнтным 2 по модулю 3. Всего было предложено 42 числа, по одному числу в диапазоне от 100 до 500 разрядов с шагом 10 разрядов (плюс одно дополнительное, 129-разрядное число). На данный момент RSA-100, RSA-110, RSA-120, и RSA-129 были разложены на множители, все с помощью QS. Следующим (с помощью NFS) может быть RSA-130, или чемпионы по разложению на множители сразу возьмутся за RSA -140.

Данная область развивается быстро. Технику разложения на множители трудно экстраполировать, так как невозможно предсказать развитие математической теории. До

открытия NFS многие считали, что любой метод разложения на множители не может асимптотически быть быстрее QS. Они были неправы.

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

Квадратные корни по модулю n

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

2.12 Генерация простого числа

Для алгоритмов с открытыми ключами необходимы простые числа. Их нужно множество для любой достаточно большой сети. Прежде, чем обсуждать математику генерации простого числа, рассмотрим некоторые вопросы.

Генерация случайных чисел с последующей попыткой разложения их на множители - это неправильный способ поиска простых чисел. Существуют различные вероятностные проверки на простоту чисел, определяющие, является ли число простым, с заданной степенью достоверности. При условии, что эта "степень достоверности" достаточно велика, такие способы проверки достаточно хороши.

Предположим, что одна проверка из 250 - ошибочна. Это означает, что с вероятностью 1/10 15 проверка объявит простым составное число. Простое число никогда не будет объявлено составным при проверке. Если по какой-то причине понадобится большая достоверность простоты числа, уровень ошибки можно понизить. Рассмотрим несколько вероятностных тестов на простоту.

Тест Solovay-Strassena

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

Выберите случайно число a, меньшее p.

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

Вычислите $j = a^{(p-1)/2} \bmod p$

Вычислите символ Якоби J(a,p).

Если j ≠J(a,p), то число p наверняка не является простым.

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

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

Тест Lehmanna

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

1. Выберите случайно число a, меньшее p.
2. Вычислите $a^{(p-1)/2}$ mod p
3. Если $a^{(p-1)/2} \neq 1$ или -1 (mod p), то p не является простым.
4. Если $a^{(p-1)/2} = 1$ или -1 (mod p), то вероятность того, что число p не является простым, не больше 50 процентов.

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

Тест Rabin-Millera

Повсеместно используемым является простой алгоритм, разработанный Майклом Рабином (Michael Rabin), частично основанным на идеях Гэри Миллера.

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

1. Выберите случайное число a, меньшее p.
2. Установите j = 0 и z = am mod p
3. Если z = 1 или если z = p - 1, то p проходит проверку и может быть простым числом.
4. Если j > 0 и z = 1, то p не является простым числом.
5. Установите j = j + 1. Если j < b и $z \neq p - 1$, установите $z = z^2$ mod p и вернитесь на этап (4). Если z = p - 1, то p проходит проверку и может быть простым числом.
6. Если j = b и $z \neq p - 1$, то p не является простым числом.

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

Существуют более точные оценки. Для n-битового кандидата в простые числа (где n больше 100), вероятность ошибки в одном тесте меньше, чем $4^{n2(k/2)1}/2$. И для 256-битового n вероятность ошибки в шести тестах меньше, чем 1/251.

Практические соображения

В реальных приложениях генерация простых чисел происходит быстро.

1. Сгенерируйте случайное n-битовое число p
2. Установите старший и младший биты равными 1. (Старший бит гарантирует требуемую длину простого числа, а младший бит обеспечивает его нечетность.)
3. Убедитесь, что p не делится на небольшие простые числа: 3, 5, 7, 11, и т.д. Во многих реализациях проверяется делимость p на все простые числа, меньшие 256. Наиболее эффективной является проверка на делимость для всех простых чисел, меньших 2000.
4. Выполните тест Rabin-Miller для некоторого случайного a. Если p проходит тест, сгенерируйте другое случайное a и повторите проверку. Выбирайте небольшие значения a для ускорения вычислений. Выполните пять тестов. (Одного может показаться достаточным, но выполните пять.) Если p не проходит одной из проверок, сгенерируйте другое p и попробуйте снова.

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

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

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

Сильные простые числа

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

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

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

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

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

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

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

Есть много споров необходимо ли генерировать сильно простых чисел. Длина простых чисел гораздо важнее их структуры. Более того, сама структура уменьшает случайность числа и может снизить устойчивость системы.

Но могут быть созданы новые методы разложения на множители, которые лучше работают с числами, обладающими определенными свойствами. В этом случае снова могут потребоваться сильные простые числа.

2.13 Дискретные логарифмы в конечном поле

В качестве другой однонаправленной функции в криптографии часто используется возведение в степень по модулю. Легко вычислить: $a^x \bmod n$.

Задачей, обратной возведению в степень по модулю, является поиск дискретного логарифма. Задача состоит в нахождении x, для которого $a^x = b \pmod n$.

Например:

Если $3^x = 15 \bmod 17$, то $x = 6$

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

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

Еще сложнее решать эту задачу для 1024-битовых чисел.

Вычисление дискретных логарифмов в конечной группе

Криптографы интересуются дискретными логарифмами следующих трех групп:

Мультипликативная группа полей простых чисел: GF(p)

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

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

Безопасность многих алгоритмов с открытыми ключами основана на задаче поиска дискретных логарифмов, поэтому эта задача была глубоко изучена.
Если p является простым числом и используется в качестве модуля, то сложность поиска дискретных логарифмов в GF(p) соответствует разложению на множители числа n того же размера, где n - это произведение двух простых чисел приблизительно равной длины. То есть:
$e^{(1+O(1))(\ln(n))1/2(\ln((\ln(n)))1/2}$

Решето числового поля быстрее, оценка его эвристического времени выполнения:
$e^{(1.923+O(1))(\ln(n))1/3(\ln((\ln(n)))2/3}$

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

Вычисление дискретных логарифмов тесно связано с разложением на множители. Если вы можете решить проблему дискретного логарифма, то вы можете и разложить на множители. Истинность обратного никогда не была доказана. В настоящее время существует три метода вычисления дискретных логарифмов в поле простого числа: линейное решето, схема целых чисел Гаусса и решето числового поля.
Предварительное, объемное вычисление для поля должно быть выполнено только один раз. Затем, быстро можно вычислять отдельные логарифмы. Это может серьезно уменьшить безопасность систем, основанных на таких полях. Важно, чтобы различные приложения использовали различные поля простых чисел. Хотя несколько пользователей одного приложения могут применять общее поле.

В мире расширенных полей исследователями не игнорируются и $GF(2^n)$. Алгоритм Копперсмита (Coppersmith) позволяет за приемлемое время находить дискретные логарифмы в таких полях как $GF(2^{127})$ и делает принципиально возможным их поиск в полях порядка $GF(2^{400})$. У этого алгоритма очень велика стадия предварительных вычислений, но во всем остальном он хорош и эффективен. Реализация менее эффективной версии этого же алгоритма после семи часов предварительных вычислений тратила на нахождение каждого дискретного логарифма в поле $GF(2^{127})$ лишь несколько секунд. Это конкретное поле, когда-то использовавшееся в некоторых криптосистемах, не является безопасным.

Позднее были выполнены предварительные вычисления для полей $GF(2^{227})$, $GF(2^{313})$ и GF(2401), удалось значительно продвинуться и для поля $GF(2^{503})$. Эти вычисления проводились на nCube-2, массивном параллельном компьютере с 1024 процессорами. Вычисление дискретных логарифмов в поле $GF(2^{593})$ все еще находится за пределами возможного.

Как и для нахождения дискретных логарифмов в поле простого числа, для вычисления дискретных логарифмов в полиномиальном поле также требуется один раз выполнить предварительные вычисления.

Раздел 3. Криптоалгоритмы

3.1 Подстановочные и перестановочные шифры

До появления компьютеров криптография состояла из алгоритмов на символьной основе. Различные криптографические алгоритмы либо заменяли одни символы другими, либо переставляли символы. Лучшие алгоритмы делали и то, и другое, и по много раз.

Сегодня все значительно сложнее, но философия остается прежней. Первое изменение заключается в том, что алгоритмы стали работать с битами, а не символами. Это важно хотя бы с точки зрения размера алфавита - с 26 элементов до двух. Большинство хороших криптографических алгоритмов до сих пор комбинируют подстановки и перестановки.

Подстановочные шифры

Подстановочным шифром называется шифр, который каждый символ открытого текста в шифротексте заменяет другим символом. Получатель инвертирует подстановку шифротекста, восстанавливая открытый текст. В классической криптографии существует четыре типа подстановочных шифров:

— **Простой подстановочный шифр** или **моноалфавитный шифр -** шифр, который каждый символ открытого текста заменяет соответствующим символом шифротекста.

— **Однозвучный подстановочный шифр** похож на простую подстановочную криптосистему за исключением того, что один символ открытого текста отображается на несколько символов шифротекста. Например, "A" может соответствовать 5, 13, 25 или 56, "B" - 7, 19, 31 или 42 и так далее.

— **Полиграмный подстановочный шифр** - шифр, который блоки символов шифрует по группам. Например, "ABA" может соответствовать "RTQ", "ABB" может соответствовать "SLL" и так далее.

— **Полиалфавитный подстановочный шифр** состоит из нескольких простых подстановочных шифров. Например могут быть использованы пять различных простых подстановочных фильтров; каждый символ открытого текста заменяется с использованием одного конкретного шифра.

Знаменитый **шифр Цезаря,** представляет собой простой подстановочный фильтр, в котором каждый символ открытого текста заменяется символом, находящегося тремя символами правее по модулю 26 ("A" заменяется на "D," "B" - на "E", ... "W" - на " Z ", "X" - на "A", "Y" - на "B", "Z" - на "C"). Он действительно очень прост, так как алфавит шифротекста представляет собой смещенный, а не случайно распределенный алфавит открытого текста.

ROTI3 - это простая шифровальная программа, обычно поставляемая с системами UNIX. Она также является простым подстановочным шифром. В этом шифре "A" заменяется на "N," "B" - на "O" и так далее. Каждая буква смещается на 13 мест. Шифрование файла программой ROTI3 дважды восстанавливает первоначальный файл.

P = ROT13 (ROT 13 (P))

ROTI3 часто применяется в почте, закрывая потенциально неприятный текст, и не используется для безопасности.

Простые подстановочные шифры легко раскрываются, так как шифр не прячет частоты использования различных символов в открытом тексте. Чтобы восстановить открытый текст, хорошему криптоаналитику требуется знать только 26 символов английского алфавита.

Однозвучные подстановочные шифры использовались уже в 1401 году в герцогстве Мантуа. Они были более сложны для вскрытия, чем простые подстановочные шифры, хотя и они не скрывают всех статистических свойств языка открытого текста. При помощи вскрытия с известным открытым текстом эти шифры раскрываются тривиально. Вскрытие с использованием только шифротекста более трудоемко, но и оно занимает на компьютере лишь несколько секунд.

Полиграмные подстановочные шифры - шифры, которые кодируют сразу группы символов. Шифр Play- fair ("Честная игра"), изобретенный в 1854 году, использовался англичанами в Первой мировой войне, он шифрует пары символов. Другим примером полиграмного подстановочного шифра является шифр Хилла (Hill). Иногда, можно видеть, как вместо шифра используется кодирование по Хаффману (Huffman), это небезопасный полиграмный подстановочный шифр.

У полиалфавитных подстановочных шифров множественные однобуквенные ключи, каждый из которых используется для шифрования одного символа открытого текста. Первым ключом шифруется первый символ открытого текста, вторым ключом - второй символ, и так далее. После использования всех ключей они повторяются циклически. Если применяется 20 однобуквенных ключей, то каждая двадцатая буква шифруется тем же ключом. Этот параметр называется **периодом** шифра. В классической криптографии шифры с длинным периодом было труднее раскрыть, чем шифры с коротким периодом. Использование компьютеров позволяет легко раскрыть подстановочные шифры с очень длинным периодом.

Шифр с бегущим ключом (иногда называемый книжным шифром), использующий один текст для шифрования другого текста, представляет собой другой пример подобного шифра. И хотя период этого шифра равен длине текста, он также может быть легко взломан.

Перестановочные шифры

В **перестановочном шифре** меняется не открытый текст, а порядок символов. В **простом столбцовом перестановочном шифре** открытый текст пишется горизонтально на разграфленном листе бумаги фиксированной ширины, а шифротекст считывается по вертикали. Дешифрирование представляет собой запись шифротекста вертикально на листе разграфленной бумаги фиксированной ширины и затем считывание открытого текста горизонтально.

Так как символы шифротекста те же, что и в открытом тексте, частотный анализ шифротекста покажет, что каждая буква встречается приблизительно с той же частотой, что и обычно. Это даст криптоаналитику возможность применить различные методы, определяя правильный порядок символов для получения открытого текста. Применение к шифротексту второго перестановочного фильтра значительно повысит безопасность. Существуют и еще более сложные перестановочные фильтры, но компьютеры могут раскрыть почти все из них.

Немецкий шифр ADFCVX, использованный в ходе Первой мировой войны, представлял собой перестановочный фильтр в сочетании с простой подстановкой. Этот, очень сложный алгоритм для своего времени, был раскрыт Жоржем Пенвэном (Georges Painvin), французским криптоаналитиком.

Хотя многие современные алгоритмы используют перестановку, с этим связана проблема использования большого объема памяти, а также иногда требуется работа с сообщениями определенного размера. Подстановка более обычна.

Рассмотрим более подробно работу перестановочных шифров, т.к. идеи, лежащие в основе шифра замены, движут и современными криптографами, разрабатывающими симметричные алгоритмы. Позже мы увидим, что в шифрах *DES* и *Rijndael* присутствуют компоненты, называемые S-блоками, которые являются простыми подстановками.

Перестановочные шифры активно применялись в течение нескольких столетий. Мы рассмотрим один из самых простейших перестановычных шифров, который легко поддается взламыванию.

Фиксируется симметрическая группа S_n и какой-то ее элемент a принадлежит S_n..

Именно перестановка является секретным ключом.

Предположим, что

$$\sigma = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 4 & 1 & 3 & 5 \end{pmatrix} = (1243) \in S_5$$

и зашифруем с ее помощью открытый текст:

Once upon a time there was a little girl called Snow White.

Разобьем текст на части по 5 букв:

опсеи ponat imeth erewa salit egirl calle dsnow white.

Затем переставим буквы в них в соответствии с нашей перестановкой:

соепи npaot eitmh eewra Isiat iergl Iclae ndosw iwthe.

Убрав теперь промежутки между группами, чтобы скрыть значение n, получим шифротекст

coenunpaoteitmheewralsiatiergllclaendoswiwthe.

Перестановочный шифр поддается взлому атакой с выбором открытого текста в предположении, что участвующая в шифровании использованная симметрическая группа (т. е. параметр *n)* не слишком велика. Для этого нужно навязать отправителю шифрованных сообщений нужный нам открытый текст и получить его в зашифрованном виде. Предположим, например, что нам удалось подсунуть алфавит

abcdefghijklmnopqrstuvwxyz

и получить его шифрованную версию

CADBEHFIGJMKNLORPSQTWUXVYZ.

Сопоставляя открытый текст и криптограмму, получаем перестановку,

$$\begin{pmatrix} 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10\ 11\ 12\ 13\ 14\ 15 \\ 2\ 4\ 1\ 3\ 5\ 7\ 9\ 6\ 8\ 10\ 12\ 14\ 11\ 13\ 15 \end{pmatrix}$$

После короткого анализа полученной информации мы обнаруживаем повторяющиеся серии перестановок: каждые пять чисел переставляются одинаково. Таким образом, можно сделать вывод, что n=5, и восстановить ключ, взяв первые пять столбцов этой таблицы.

Роторные машины

В 1920-х годах для автоматизации процесса шифрования были изобретены различные механические устройства. Большинство использовало понятие **ротора,** механического колеса, используемого для выполнения подстановки.

Роторная машина, включающая клавиатуру и набор роторов, реализует вариант шифра Вигенера. Каждый ротор представляет собой произвольное размещение алфавита, имеет 26 позиций и выполняет простую подстановку. Например, ротор может быть использован для замены "A" на " F", "B" на "U", "C" на "I" и так далее. Выходные штыри одного ротора соединены с входными штырями следующего ротора.

Например, в четырехроторной машине первый ротор может заменять "A" на " F", второй - "F" на "Y", третий - "Y" на "E" и четвертый - "E" на "C", "C" и будет конечным шифротекстом. Затем некоторые роторы смещаются, и в следующий раз подстановки будут другими.

Именно комбинация нескольких роторов и механизмов, движущих роторами, и обеспечивает безопасность машины. Так как роторы вращаются с различной скоростью, период для n-роторной машины равен 26^n. Некоторые роторные машины также могут иметь различные положения для каждого ротора, что делает криптоанализ еще более бессмысленным.

Самым известным роторным устройство является Энигма (Enigma). Энигма использовалась немцами во Второй мировой войне. Сама идея пришла в голову Артуру Шербиусу (Arthur Scherbius) и Арвиду Герхарду Дамму (Arvid Gerhard Damm) в Европе. В Соединенных Штатах она была запатентована Артуром Шербиусом. Немцы значительно усовершенствовали базовый проект для использования во время войны.

У немецкой Энигмы было три ротора, которые можно было выбрать из пяти возможных, коммутатор, который слегка тасовал открытый текст, и отражающий ротор, который заставлял каждый ротор обрабатывать открытый текст каждого письма дважды. Несмотря на сложность Энигмы, она была взломана в течение Второй мировой войны. Сначала группа польских криптографов взломала немецкую Энигму и объяснила раскрытый алгоритм англичанам. В ходе войны немцы модифицировали Энигму, а англичане продолжали криптоанализ новых версий.

Простое XOR

XOR представляет собой операцию "исключающее или": ‘^’ в языке C или Q в математической нотации. Это обычная операция над битами:

$$0 \oplus 0 = 0$$
$$0 \oplus 1 = 1$$
$$1 \oplus 0 = 1$$
$$1 \oplus 1 = 0$$

Также заметим, что:

$$a \oplus a = 0$$
$$a \oplus b \oplus b = a$$

Казалось бы, запутанный алгоритм простого XOR по сути является ничем иным, как полиалфавитным шифром Вигенера. Здесь он упоминается только из-за распространенности в коммерческих программных продуктах, по крайней мере в мире MS-DOS и Macintosh.

Рассмотри алгоритм, использующий только операцию XOR и покажем, что такой способ шифрования небезопасен.

Предположим, что открытый текст использует английский язык. Более того, пусть длина ключа любое небольшое число байт. Ниже описано, как взломать этот шифр:

1. Определим длину ключа с помощью процедуры, известной как **подсчет совпадений**. Применим операцию XOR к шифротексту, используя в качестве ключа сам шифротекст с различными смещениями, и подсчитаем совпадающие байты. Если величина смещения кратна длине ключа, то совпадет свыше 6 процентов байтов. Если нет, то будут совпадать меньше чем 0.4 процента (считая, что обычный ASCII текст кодируется случайным ключом, для других типов открытых текстов числа будут другими). Это называется **показателем совпадений.** Минимальное смещение от одного значения, кратного длине ключа, к другому и есть длина ключа.
2. Сместим шифротекст на эту длину и проведем операцию XOR для смещенного и оригинального шифротекстов. Результатом операции будет удаления ключа и получение открытого текста, подвергнутого операции XOR с самим собой, смещенным на длину ключа. Так как в английском языке на один байт приходится 1.3 бита действительной информации, существующая значительная избыточность позволяет определить способ шифрования.

Одноразовые блокноты

Идеальный способ шифрования существует. Он называется **одноразовым блокнотом** и был изобретен в 1917 году Мэйджором Джозефом Моборном (Major Joseph Mauborgne) и Гилбертом Вернамом (Gilbert Vernam) из AT&T. (Фактически одноразовый блокнот представляет собой особый случай пороговой схемы.) В классическом понимании одноразовый блокнот является большой неповторяющейся последовательностью символов ключа, распределенных случайным образом, написанных на кусочках бумаги и приклеенных к листу блокнота. Первоначально это была одноразовая лента для телетайпов. Отправитель использовал каждый символ ключа блокнота для шифрования только одного символа открытого текста. Шифрование представляет собой сложение по модулю 26 символа открытого текста и символа ключа из одноразового блокнота.

Каждый символ ключа используется только единожды и для единственного сообщения. Отправитель шифрует сообщения и уничтожает использованные страницы блокнота или использованную часть ленты. Получатель, в свою очередь, используя точно такой же блокнот, дешифрирует каждый символ шифротекста. Расшифровав сообщение, получатель уничтожает соответствующие страницы блокнота или часть ленты . Новое сообщение - новые символы ключа. Например, если сообщением является:

ONETIMEPAD а ключевая последовательность в блокноте:

TBFRGFARFM то шифротекст будет выглядеть как:

IPKLPSFHGQ так как

$$Q + T \bmod 26 = I$$
$$N + B \bmod 26 = P$$
$$E + F \bmod 26 = K$$

и т.д.

В предположении, что злоумышленник не сможет получить доступ к одноразовому блокноту, использованному для шифрования сообщения, эта схема совершенно безопасна. Данное шифрованное сообщение на вид соответствует любому открытому сообщению того же размера.

Так как все ключевые последовательности совершенно одинаковы (помните, символы ключа генерируются случайным образом), у противника отсутствует информация, позволяющая подвергнуть шифротекст криптоанализу. Кусочек шифротекста может быть похож на:

POYYAEAAZX что
дешифрируется как:
SALMONEGGS или
на:
BXEGBMTMXM что
дешифрируется как:
GREENFLUID

Так как все открытые тексты равновероятны, у криптоаналитика нет возможности определить, какой из открытых текстов является правильным. Случайная ключевая последовательность, сложенная с неслучайным открытым текстом, дает совершенно случайный шифротекст, и никакие вычислительные мощности не смогут это изменить.

Необходимо напомнить, что символы ключа должны генерироваться случайным образом. Любые попытки вскрыть такую схему сталкиваются со способом, которым создается последовательность символов ключа. Использование генераторов псевдослучайных чисел не считается, у них всегда неслучайные свойства.

Другой важный момент: ключевую последовательность никогда нельзя использовать второй раз. Даже если вы используете блокнот размером в несколько гигабайт, то если криптоаналитик получит несколько текстов с перекрывающимися ключами, он сможет восстановить открытый текст. Он сдвинет каждую пару шифротекстов относительно друг друга и подсчитает число совпадений в каждой позиции. Если шифротексты смещены правильно, соотношение совпадений резко возрастет - точное значение зависит от языка открытого текста. С этой точки зрения криптоанализ не представляет труда. Это похоже на показатель совпадений, но сравниваются два различных "периода".

Идея одноразового блокнота легко расширяется на двоичные данные. Вместо одноразового блокнота, состоящего из букв, используется одноразовый блокнот из битов . Вместо сложения открытого текста с ключом одноразового блокнота используйте XOR. Для дешифрирования примените XOR к шифротексту с тем же одноразовым блокнотом. Все остальное не меняется, и безопасность остается такой же совершенной.

Все это хорошо, но существует несколько проблем. Так как ключевые биты должны быть случайными и не могут использоваться снова, длина ключевой последовательности должна равняться длине сообщения. Одноразовый блокнот удобен для нескольких небольших сообщений, но его нельзя использовать для работы по каналу связи с пропускной способностью 1.544 Мбит/с. Вы можете хранить 650 Мбайт случайных данных на CD-ROM, но и тут есть проблемы. Во первых, вам нужно только две копии случайных битов, но CD-ROM экономичны только при больших тиражах. И во вторых, вам нужно уничтожать использованные биты. Для CD-ROM нет другой возможности удалить информацию кроме как физически разрушить весь диск. Гораздо больше подходит цифровая лента.

Даже если проблемы распределения и хранения ключей решены, вам придется точно синхронизировать работу отправителя и получателя. Если получатель пропустит бит (или несколько бит пропадут при передаче), сообщение потеряет всякий смысл. С другой стороны, если несколько бит изменятся при передаче (и ни один бит не будет удален или добавлен - что гораздо больше похоже на влияние случайного шума), то лишь эти биты будут расшифрованы неправильно. Но одноразовый блокнот не обеспечивает проверку подлинности.

Одноразовые блокноты используются и сегодня, главным образом для сверхсекретных каналов связи с низкой пропускной способностью.

3.2 Компьютерные алгоритмы

Существует множество компьютерных алгоритмов. Следующие четыре используются чаще всего:

— DES (Data Encryption Standard, стандарт шифрования данных) - популярный компьютерный алгоритм шифрования, являлся американским и международным стандартом. Это симметричный алгоритм, один и тот же ключ используется для шифрования и дешифрирования.

— AES (Advanced Encryption Standart, стандарт шифрования данных) – пришел на смену алгоритму DSA, был разработан бельгийскими криптографами, в его основе лежит алгоритм Rijndael.

— RSA (назван в честь создателей - Ривеста (Rivest), Шамира (Sharnir) и Эдлмана (Adleman)) - самый популярный алгоритм с открытым ключом. Используется и для шифрования, и для цифровой подписи.

— DSA (Digital Signature Algorithm, алгоритм цифровой подписи, используется как часть стандарта цифровой подписи, Digital Signature Standard) - другой алгоритм с открытым ключом. Используется только для цифровой подписи, не может быть использован для шифрования.

Стандарт шифрования данных DES (Data Encryption Standard)

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

Покупатели не знали, что они покупают. Многие небольшие компании изготавливали и продавали криптографическое оборудование, преимущественно заокеанским правительствам. Все это оборудование отличалось друг от друга и не могло взаимодействовать. Никто не знал, действительно ли какое-либо из этих устройств безопасно, не существовало независимой организации, которая засвидетельствовала бы безопасность.

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

15 мая 1973 года в Federa/ Register NBS опубликовало требования к криптографическому алгоритму, который мог бы быть принят в качестве стандарта. Было приведено несколько критериев оценки проекта:

- Алгоритм должен обеспечивать высокий уровень безопасности.
- Алгоритм должен быть полностью определен и легко понятен.
- Безопасность алгоритма должна основываться на ключе и не должна зависеть от сохранения в тайне самого алгоритма.
- Алгоритм должен быть доступен всем пользователям.
- Алгоритм должен позволять адаптацию к различным применениям.
- Алгоритм должен позволять экономичную реализацию в виде электронных приборов.

- Алгоритм должен быть эффективным в использовании.
- Алгоритм должен предоставлять возможности проверки.
- Алгоритм должен быть разрешен для экспорта.

Реакция общественности показала, что к криптографическому стандарту существует заметный интерес, но опыт в этой области чрезвычайно мал. Ни одно из предложений не удовлетворяло предъявленным требованиям.
27 августа 1972 года в Federal Register NBS опубликовало повторное предложение. Наконец, появился подходящий кандидат: алгоритм под именем Люцифер, в основе которого лежала разработка компании IBM, выполненная в начале 70-х.

Несмотря на определенную сложность алгоритм был прямолинеен. Он использовал только простые логические операции над небольшими группами битов и мог быть довольно эффективно реализован в аппаратуре.

Стандарт шифрования данных DES 23 ноября 1976 года был принят в качестве федерального стандарта и разрешен к использованию на всех несекретных правительственных коммуникациях.

Описание DES

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

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

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

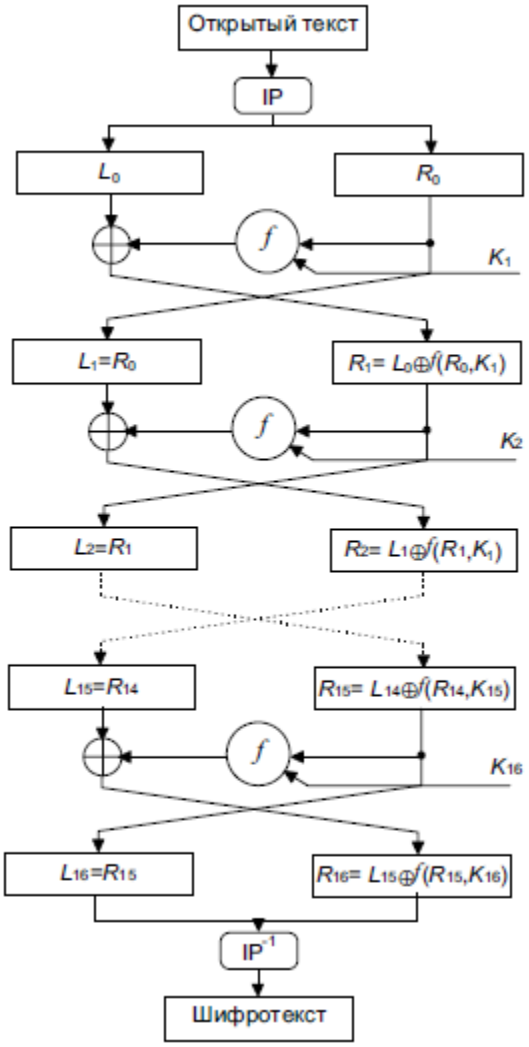


Рис 3.1 DES.

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

Схема алгоритма

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

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

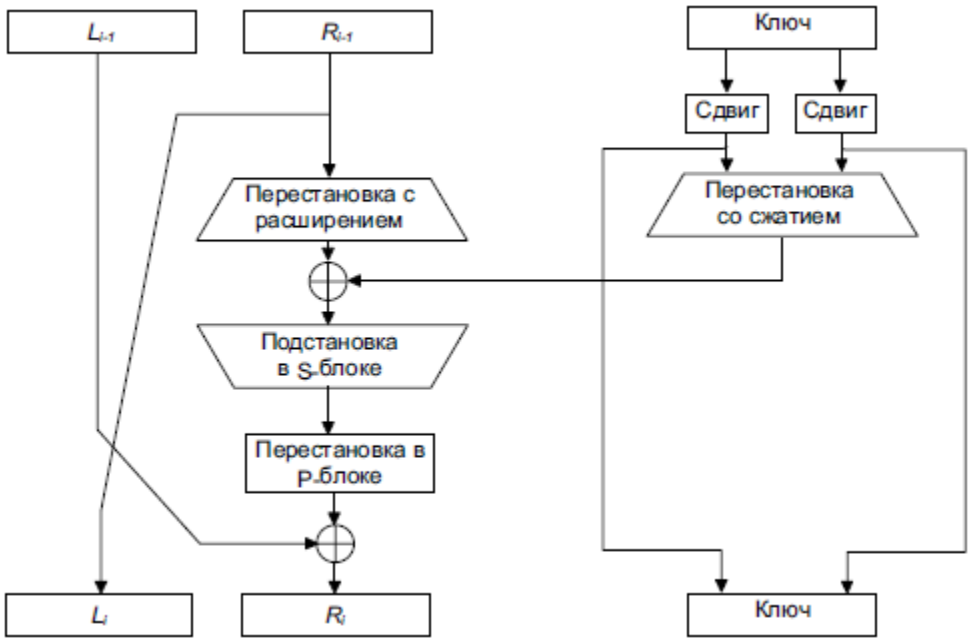


Рис. 3.2 Один этап DES.

Начальная перестановка

Табл. 3.1 Начальная перестановка

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

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

Начальная перестановка и соответствующая заключительная перестановка не влияют на безопасность DES. Эта перестановка в первую очередь служит для облегчения побайтной загрузки данных открытого текста и шифротекста в микросхему DES. Не забывайте, что DES появился раньше 16- и 32- битовых микропроцессорных шин.

Преобразования ключа

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

Табл. 3.2 Перестановка ключа

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

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

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

Табл. 3.3. Перестановка со сжатием

14,	17,	11,	2,4,	1,	5,	3,	28,	15,	6,	21,	10,
23,	19,	11,	4,	26,	8,	16,	7,	27,	20,	13,	2,
41,	52,	31,	37,	47,	55,	30,	40,	51,	45,	33,	48,
44,	49,	39,	56,	34,	53,	46,	42,	50,	36,	29,	32

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

Перестановка с расширением

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

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

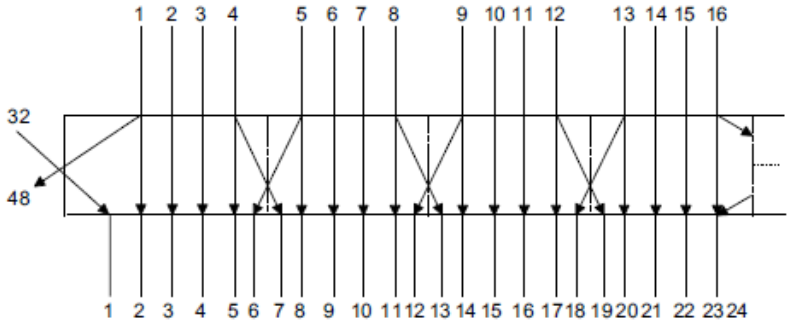


Табл. 3.4 Перестановка с расширением

32,	1,	2,	3,	4,	5,	4,	5,	6,	7,	8,	9,
8,	9,	10,	11,	12.,	13,	12,	13,	14,	15,	16,	17,

16,	17,	18,	19,	20,	21,	20,	21,	22,	23,	24,	25,
24,	25,	26,	27,	28,	29,	28,	29,	30,	31,	32,	1

Подстановка с помощью S-блоков

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

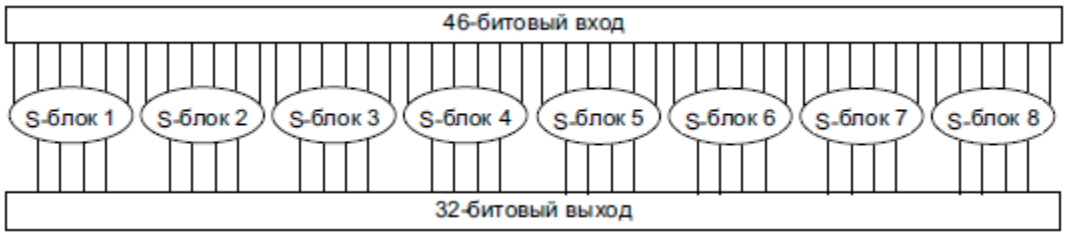


Рис. 3.3 Подстановка - S-блоки.

Каждый S-блок представляет собой таблицу из 4 строк и 16 столбцов. Каждый элемент в блоке является 4- битовым числом. По 6 входным битам S-блока определяется, под какими номерами столбцов и строк искать выходное значение. Все восемь S-блоков показаны в таблице 3.5

Табл. 3.5 S-блоки

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

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

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

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

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

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

Перестановка с помощью P-блоков

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

Табл. 3.6 Перестановка с помощью P-блоков

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

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

Заключительная перестановка

Заключительная перестановка является обратной по отношению к начальной перестановки. Обратите внимание, что левая и правая половины не меняются местами после последнего этапа DES, вместо этого объединенный блок $R_{16}L_{16}$ используется как вход заключительной перестановки. Это сделано для того, чтобы алгоритм можно было использовать как для шифрования, так и для дешифрирования.

Табл. 3.7 Заключительная перестановка

40,	8,	48,	16,	56,	24,	64,	32,	39,	7,	47,	15,	55,	23,	63,	31,

38,	6,	46,	14,	54,	22,	62,	30,	37,	5,	45,	13,	53,	21,	61,	29,
36,	4,	44,	12,	52,	20,	60,	28,	35,	3,	43,	11,	51,	19,	59,	27,
34,	2,	42,	10,	50,	18,	58,	26,	33,	1,	41,	9,	49,	17,	57,	25

Дешифрирование DES

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

DES позволяет использовать для шифрования или дешифрирования блока одну и ту же функцию. Единственное отличие состоит в том, что ключи должны использоваться в обратном порядке. То есть, если на этапах шифрования использовались ключи K_1, K_2, K_3, ..., K_{16}, то ключами дешифрирования будут K_{16}, K_{15}, K_{14}, ..., K_1.

Advanced Encryption Standard (AES)

Advanced Encryption Standard (AES), также известный как Rijndael— симметричный алгоритм блочного, принятый в качестве стандарта шифрования правительством США по результатам конкурса AES .

Для обеспечения криптостойкости алгоритм *Rijndael* включает в себя повторяющиеся раунды, каждый из которых состоит из замен, перестановок и прибавления ключа. Кроме того, *Rijndael* использует сильную математическую структуру: большинство его операций основаны на арифметике поля F_2^8. Однако, в отличие от *DES*, шифрование и расшифрование в этом алгоритме — процедуры разные.

Напомним, что элементы поля F_2^8 хранятся в памяти компьютера в виде 8-битовых векторов (или байтов), представляющих двоичные многочлены. Например, байт '83h’ в шестнадцатиричной системе соответствует двоичному числу

'1000 00116', т. к. '83h’ = 8 - 16 + 3 = 131

в десятичной системе. Этот набор двоичных разрядов можно получить непосредственно из байта '83h' = '80h’ + '03h', заметив, что цифра 8 шестнадцатиричной системы, стоящая во втором разряде представляет число $8 \cdot 16 = 8 \cdot 2^4$. Само число $8 = 2^3$ в двоичной системе записывается в виде последовательности 1000 двоичных знаков. Значит, число '80h’ = $2^3 \bullet 2^4$ в двоичной системе счисления выглядит как '1000 0000b'. Осталось к этой строке битов прибавить запись числа 3 в двоичной системе, т.е. 0011. Заметьте, что при этом достаточно к числу 8 (1000) в двоичной системе приписать число 3 (0011). Указанная последовательность битов соответствует многочлену $x^7 + x + 1$ над полем F_2. Таким образом, можно сказать, что шестнадцатиричное число '83h’ представляет тот же многочлен.

Арифметические операции в поле F_2^8 соответствуют операциям над двоичными многочленами из $F_2[X]$ по модулю неприводимого полинома

$$m(X) = X^8 + X^4 + X^3 + X + 1.$$

В алгоритме *Rijndael* 32-битовые слова отождествляются с многочленами степени 3 из $F_2^8[X]$. Отождествление делается в формате «перевертыш», т.е. старший (наиболее значимый) бит соответствует младшему коэффициенту многочлена. Так, например, слово

$$a_o||a_1||a_2||a_3$$

соответствует многочлену

$$a_3X^3 + a_2X^2 + a_1X + a_0.$$

Арифметика в алгоритме совпадает с арифметическими действиями в кольце многочленов $F_{28}[X]$ по модулю многочлена $M(X) = X^4 + 1$. Заметим, что многочлен $M(X) = (X + 1)^4$ приводим, и, следовательно, арифметические действия в алгоритме отличны от операций поля, в частности, бывают пары ненулевых элементов, произведение которых равно 0.

Rijndael — настраиваемый блочный алгоритм, который может работать с блоками из 128, 192 или 256 битов. Для каждой комбинации блока и размера ключа определено свое количество раундов. В целях упрощения обсуждения мы рассмотрим самый простой и, вероятно, наиболее часто используемый вариант алгоритма, при котором блоки, как и ключ, состоят из 128 битов. В этом случае в алгоритме выполняется 10 раундов. С данного момента мы будем иметь дело лишь с этой простой версией.

Rijndael оперирует с внутренней байтовой матрицей размера 4×4, называемой матрицей состояний:

$$S = \begin{pmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{pmatrix},$$

которую обычно записывают как вектор 32-битовых слов. Каждое слово в векторе представляет столбец матрицы. Подключи также хранятся в виде матрицы 4×4:

$$K_i = \begin{pmatrix} k_{0,0} & k_{0,1} & k_{0,2} & k_{0,3} \\ k_{1,0} & k_{1,1} & k_{1,2} & k_{1,3} \\ k_{2,0} & k_{2,1} & k_{2,2} & k_{2,3} \\ k_{3,0} & k_{3,1} & k_{3,2} & k_{3,3} \end{pmatrix}.$$

Операции алгоритма Rijndael

Раундовая функция в *Rijndael* действует с использованием четырех операций, которые мы сейчас опишем.

SubBytes.

В алгоритме есть два типа S-блоков. Один тип применяется при шифровании, а другой — при расшифровании.

Каждый из них обратен другому. S-блоки в алгоритме *DES* отбирались из большого числа себе подобных так, чтобы предотвратить взлом шифра с помощью дифференциального криптоанализа. В *Rijndael* S-блоки имеют прозрачную математическую структуру, что позволяет формально анализировать устойчивость шифра к дифференциальному и линейному анализам. Эта математическая структура не только повышает сопротивляемость дифференциальному анализу, но и убеждает пользователя, что в алгоритм не закрался недосмотр.

S-блоки поочередно обрабатывают строки матрицы состояний $s = [s_7,...,s_0]$, воспринимая их как элементы поля F_2^8. Их работа состоит из двух шагов.

1. Вычисляется мультипликативный обратный к элементу s принадлежащий F_{28} и записывается как новый байт x = [x_7, ..., x_0]. По соглашению, элемент [0, ..., 0], не имеющий обратного, остается неизменным.
2. Битовый вектор x с помощью линейного преобразования над полем F_2 переводится в вектор *y:*

$$\begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix},$$

служащий выходом S-блока. Действия S-блока на стадии расшифрования состоят в обратном линейном преобразовании и вычислении мультипликативного обратного. Эти преобразования байтов можно осуществить, используя табличный поиск или микросхему, реализующую вычисление обратных элементов в F_2^8 и линейные преобразования.

ShiftRows.

Операция ShiftRows в *Rijndael* осуществляет раундический сдвиг матрицы состояний. Каждая из ее строк сдвигается на свое число позиций. В рассматриваемой версии шифра это преобразование имеет вид:

$$\begin{pmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{pmatrix} \mapsto \begin{pmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,1} & s_{1,2} & s_{1,3} & s_{1,0} \\ s_{2,2} & s_{2,3} & s_{2,0} & s_{2,1} \\ s_{3,3} & s_{3,0} & s_{3,1} & s_{3,2} \end{pmatrix}.$$

Обратная операция — тоже простой циклический сдвиг, но в противоположном направлении. Операция ShiftRows гарантирует, что столбцы матрицы состояний будут «взаимодействовать» друг с другом на протяжении нескольких раундов.

MixColumns.

Операция MixColumns задумана с тем, чтобы строки матрицы состояний «взаимодействовали» друг с другом на протяжении всех раундов. В комбинации с предыдущей операцией она наделяет каждый байт выходных данных зависимостью от каждого байта на входе.

Мы представляем каждый столбец матрицы состояний как многочлен степени 3 с коэффициентами из F_2^8:

$$a(X) = a_0 + a_1X + a_2X^2 + a_3X^3.$$

Новый столбец получается умножением многочлена $a(X)$ на фиксированный многочлен

$c(x)$ = '02h' + '01h' • X + '01 h' • X^2 + '03h' • X^3

по модулю многочлена M(X) = X^4 + 1. Так как умножение на многочлен — линейная операция, ее можно представить в виде действия матрицы:

$$\begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix} = \begin{pmatrix} '02h' & '03h' & '01h' & '01h' \\ '01h' & '02h' & '03h' & '01h' \\ '01h' & '01h' & '02h' & '03h' \\ '03h' & '01h' & '01h' & '02h' \end{pmatrix} \cdot \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix}$$

Матрица коэффициентов невырождена над F_2s, поэтому операция MixColumns обратима, а обратное к ней действие реализуется матрицей, обратной к выписанной.

AddRoundkey.

Сложение с подключом осуществляется просто. Нужно сложить по модулю 2 все байты матрицы состояний с соответствующими элементами матрицы подключа.

Структура раундов

Запишем алгоритм *Rijndael* на псевдокоде.

```
AddRoundKey(S,K [0]);
for (i=l; i<=9; i++)
{ SubBytes(S);
  ShiftRows(S);
  MixColumns(S);
  AddRoundKey(S,K[i]);
    }
SubBytes(S);
ShiftRows(S);
AddRoundKey(S,K[10])
```

Блок открытого текста, предназначенный для шифрования, записывается в виде матрицы состояний *S.* Полученный в результате алгоритма шифротекст представляется той же матрицей. Обратите внимание, что в последнем раунде операция MixColumns не осуществляется.

Процедура расшифрования представлена следующей программой на псевдокоде.

```
    AddRoundKey(S,K[10] );
InverseShiftRows(S);
InverseSubBytes(S);
 for (i=9; i>=l; i — )
    { AddRoundKey(S,K[i] );
      InverseMixColumns(S);
      InverseShiftRows(S);
      InverseSubBytes(S)
                                                    }
AddRoundKey(S,K[0]);
```

Алгоритм RSA

Алгоритм *RSA*, первый из алгоритмов шифрования с открытым ключом, достойно выдержал испытание временем. Этот алгоритм основывается на задаче *RSA*, она сводится к поиску простых делителей больших натуральных чисел. Так что можно утверждать, что

криптостойкость алгоритма *RSA* базируется на сложности проблемы факторизации, хотя и не в полной мере, поскольку задачу *RSA* можно решать, не прибегая к разложению на множители.

Предположим, Алиса считает нужным разрешить всем желающим отправлять ей секретные сообщения, расшифровать которые способна только она. Тогда Алиса подбирает два больших простых числа p и q. Держа их в секрете, Алиса публикует их произведение N=p*q, которое называют *модулем* алгоритма.

Кроме того, Алиса выбирает шифрующую экспоненту E, удовлетворяющую условию

$$НОД\ (E,(p-1)(q-1)) = 1.$$

Как правило E берут равным 3, 17 или 65 537. Пара, доступная всем желающим, — это (N,E). Для выбора секретного ключа Алиса применяет расширенный алгоритм Евклида к паре чисел E и (p-1)(q-1), получая при этом расшифровывающую экспоненту *d*. Найденная экспонента удовлетворяет соотношению

$$E * d = 1 \pmod{(p-1)(q-1)}.$$

Секретным ключом является тройка (d,p,q). Фактически, можно было бы выбросить простые делители *p* и *q* из ключа и помнить лишь о d и всем числе N. Но это снизит скорость алгоритма.

Допустим теперь, что Боб намерен зашифровать сообщение, адресованное Алисе. Он сверяется с открытым ключом и представляет сообщение в виде числа M, строго меньшего модуля *N* алгоритма. Шифротекст *C* получается из M по следующему правилу:

$$C = M^E \pmod{N}.$$

Алиса, получив шифрограмму, расшифровывает ее, возводя число *C* в степень *d:*

$$M = C^d \pmod{N}.$$

Равенство имеет место в связи с тем, что порядок группы (Z/NZ)* равен

$$\Phi(N) = (p-1)(q-1).$$

Поэтому, по теореме Лагранжа,

$$x^{(p-1)(q-1)} = 1 \pmod{N}$$

для любого числа x принадлежит (Z/NZ)*. Поскольку E n d взаимно обратны по модулю (p-1)(q-1), при некотором целом числе *s* получается равенство

$$Ed-s(p-1)(q-1) = 1.$$

Следовательно,

$$C^d = (m^E)^d = m^{Ed} = m^{1+s(p-1)(q-1)} =$$

$$= m \cdot m^{s(p-1)(q-1)} = m \pmod{N}.$$

Для прояснения ситуации рассмотрим пример.

Пусть p = 7 и *q* = 11.

Тогда N = 77, а, (p-1)(q-1) = 6 • 10 = 60.

В качестве открытой шифрующей экспоненты возьмем число E = 37, поскольку НОД (37,60) = 1. Применяя расширенный алгоритм Евклида, найдем *d* = 13, т. к.

$$37\text{-}13 = 481 = 1 \pmod{60}.$$

Предположим, нужно зашифровать сообщение, численное представление которого имеет вид: M = 2. Тогда мы вычисляем

$$C = m^E \pmod{N} = 2^{37} \pmod{77} = 51.$$

Процесс расшифровывания происходит аналогично:

$$m = C^d \pmod{N} = 51^{13} \pmod{77} = 2.$$

Шифрование* RSA *и одноименная задача

При первом знакомстве с алгоритмом *RSA* можно увидеть, что его криптостойкость обеспечивается сложностью вычисления расшифровывающей экспоненты d по открытому ключу, т. е. по известным числу N (модулю системы) и шифрующей экспоненте.

Мы показали, что задача *RSA* не сложнее проблемы факторизации. Поэтому, если разложим на множители число N, т. е. найдем p и q, то сможем вычислить *d.* Следовательно, если проблема факторизации модуля системы окажется легкой, *RSA* будет взломана. Самые большие числа, которые к настоящему времени удается разложить на множители за разумное время, имеют 500 двоичных знаков. В связи с этим, для обеспечения стойкости систем среднего срока действия, рекомендуют брать модули шифрования порядка 1024 битов. Для систем большого срока действия следует выбирать модули, состоящие из 2048 битов.

Под криптостойкостью системы будем сейчас понимать невозможность дешифрования сообщения без знания секретного ключа, но такой подход к безопасности слишком наивен во многих приложениях. Кроме того, алгоритм *RSA* в том виде, в котором мы его представили, не может устоять против атак с выбором шифротекста.

В криптосистемах с открытым ключом атакующий всегда имеет возможность шифровать свои сообщения. Значит, он способен применять атаку с выбором открытого текста. *RSA* криптостоек против таких атак, если принять во внимание наше слабое определение стойкости и верить в трудность решения задачи *RSA.* Для аргументации этого заявления нужно использовать прием сведения одной задачи к другой. В данной ситуации доказательство безопасности алгоритма *RSA* довольно тривиально, но мы рассмотрим этот вопрос подробно.

Лемма 3. Если задача *RSA* является трудноразрешимой, то криптосистема *RSA* вычислительно защищена от атак с выбором открытого текста в том смысле, что атакующий не в состоянии корректно восстановить открытый текст, имея лишь шифротекст.

Доказательство. Разработаем алгоритм решения задачи *RSA,* основываясь на алгоритме взлома одноименной криптосистемы. Сделав это, мы покажем, что взлом криптосистемы не сложнее задачи *RSA.*

Напомним, что в задаче *RSA* дано число N, имеющее два неизвестных простых делителя p *и* q, элементы E, Y принадлежит $(Z/NZ)^*$ и требуется найти такой элемент x, что $x^E \pmod N = Y$. С помощью оракула дешифруем сообщение $C = Y$ и получим соответствующий открытый текст M, удовлетворяющий, по определению, соотношению

$$M^E \pmod N = C = Y.$$

Отсюда видно, что взломав криптосистему, мы сможем решить задачу *RSA.* ◄

Алгоритм цифровой подписи (DIGITAL SIGNATURE ALGORITHM, DSA)

В августе 19991 года Национальный институт стандартов и техники (National Institute of Standards and Technology, NIST) предложил для использования в своем Стандарте цифровой подписи (Digital Signature Standard, DSS) Алгоритм цифровой подписи (Digital Signature Algorithm, DSA).

Среди факторов, рассмотренных в процессе принятия решения были уровень обеспечиваемой безопасности, простота аппаратной и программной реализации, простота экспорта за пределы США, применимость патентов, влияние на национальную безопасность и обеспечение правопорядка, а также степень эффективностикак функции подписи, так и функции проверки. Казалось, что обеспечить соответствующую защиту Федеральным системам можно многими способами. Выбранный алгоритм должен был удовлетворять следующим требованиям:

- NIST ожидает, что его можно будет использовать бесплатно. Широкое использование этой технологии, обусловленной его доступностью, послужит к экономической выгоде правительства и общества.
- Выбранная технология обеспечивает эффективное использование операций подписи в приложениях, связанных с использованием интеллектуальных карточек. В этих приложениях операции подписи выполняются в слабой вычислительной среде интеллектуальных карточек, а процесс проверки реализуется в более мощной вычислительной среде, например, на персональном компьютере, в аппаратном криптографическом модуле или на компьютере-мэйнфрейме.

Прежде всего разберемся с названиями: DSA - это алгоритм, а DSS стандарт. Стандарт использует алгоритм. Алгоритм является частью стандарта.

Описание DSA

DSA, представляющий собой вариант алгоритмов подписи Schnorr и EIGamal. Алгоритм использует следующие параметры:

p = простое число длиной L битов, где L принимает значение, кратное 64, в диапазоне от 512 до 1024. (В первоначальном стандарте размер *p* был фиксирован и равен 512 битам, это вызвало множество критических замечаний, и NIST этот пункт алгоритма.)

q = 160-битовой простое число - множитель p-1.

$g = h^{(p-1)/q} \bmod p$ где h - любое число, меньшее p-1, для которого $h^{(p-1)/q} \bmod p$ больше 1.

x = число, меньшее q.

$y = g^x \bmod p$

В алгоритме также используется однонаправленная хэш-функция: H(m).

Первые три параметра, p, q и g, открыты и могут быть общими для пользователей сети. Закрытым ключом является x, а открытым - y. Чтобы подписать сообщение, m:

(1) Алиса генерирует случайное число k, меньшее q

(2) Алиса генерирует

$r = (g^k \bmod p) \bmod q$

$s = (k^{-1} (H(m) + xr)) \bmod q$

Ее подписью служат параметры r и s, она посылает их Бобу.

(3) Боб проверяет подпись, вычисляя

$w = s^{-1} \bmod q$

$u_1 = (H(m) * w) \bmod q$

$u_2 = (rw) \bmod q$

$v = ((g^{u1} * y^{u2}) \bmod p) \bmod q$

Если v = r, то подпись правильна.

Табл. 3.8 Подписи DSA

	Открытый ключ:
p	простое число длиной от 512 до 1024 битов (может использоваться группой пользователей)
q	160-битовый простой множитель p-1 (может использоваться группой пользователей)
g	$= h^{(p-1)/q} \bmod p$ где h - любое число, меньшее p-1, для которого $h^{(p-1)/q} \bmod p > 1$ (может использоваться группой пользователей)
y	$= g^x \bmod p$ (p-битовое число)
	Закрытый ключ:
x	< q (160-битовое число)

Подпись:

k выбирается случайно, меньшее q

r (подпись) = $(g^k \bmod p) \bmod q$

s (подпись) = $(k^{-1} (H(m) + xr)) \bmod q$

Проверка:

w = $s^{-1} \bmod q$

u_1 = $(H(m) * w) \bmod q$

u_2 = $(rw) \bmod q$

v = $((g^{u1} * y^{u2}) \bmod p) \bmod q$

Если v = r, то подпись правильна.

Табл. 3.9
Скорость DSA для различных длин модулей с 160-битовым показателем степени (на SPARC II)

	512 битов	768 битов	1024 бита
Подпись	0.20 с	0.43 с	0.57 с
Проверка	0.35 с	0.80 с	1.27 с

Практические реализации DSA часто можно ускорить с помощью предварительных вычислений. Обратите внимание, что значение r не зависит от сообщения. Можно создать строку случайных значений k, и затем рассчитать значения r для каждого из них. Можно также вычислить k^{-1} для каждого из этих значений k. Затем, когда приходит сообщение, можно вычислить s для заданных r и k^{-1}.

Эти предварительные вычисления заметно ускоряют DSA.

Раздел 4. Управление ключами и криптоанализ

4.1 Управление ключами

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

Криптоаналитики часто вскрывают и симметричные криптосистемы, и криптосистемы с открытыми ключами через распределение ключей. Зачем Еве беспокоиться о проблеме вскрытия криптографического алгоритма целиком, если она может восстановить ключ из-за неаккуратного хранения ключа?

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

Например, программа DiskLock для Macintosh (версия 2.1), продававшаяся в большинстве магазинов программного обеспечения, претендует на безопасное шифрование DES. Она шифрует файлы, используя DES. Реализация DES алгоритма правильна. Однако, DiskLock сохраняет ключ DES вместе с зашифрованным файлом. Если вы знаете, где искать ключ, и хотите прочитать файл, шифрованный DiskLock с помощью DES - восстановите ключ из шифрованного файла и затем расшифруйте файл. Не имеет значение, что программа использует шифрование DES - реализация абсолютно небезопасна.

Генерация ключей

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

Уменьшенные пространства ключей

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

В таблицах представлено число возможных ключей для различных ограничений на входные строки и приведено время, требуемое для исчерпывающего перебора всех возможных ключей при миллионе попыток в секунду.

Могут быть использованы для вскрытия грубой силой любые специализированные аппаратные и параллельные реализации. При проверке миллиона ключей в секунду (одной машиной или несколькими параллельно) физически возможно расколоть ключи из символов нижнего регистра и ключи из цифр и символов нижнего регистра длиной до 8 байтов, алфавитно-цифровые ключи - длиной до 7 байтов, ключи из печатаемых символов и ASCII-символов - длиной до 6 байтов, в ключи из 8-битовых ASCII-символов - длиной до 5 байтов.

Табл. 4.1
Количество возможных ключей в различных пространствах ключей

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

Табл. 4.2
Время исчерпывающего поиска различных пространстов ключей (при одном миллионе проверок в секунду**)**

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

Вычислительная мощь удваивается каждые 18 месяцев. Если вы хотите, чтобы ваши ключи были устойчивы к вскрытию грубой силой в течение 10 лет, вы должны соответствующим образом планировать использование ключей.

Обедненный выбор ключей

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

Это называется **вскрытием со словарем,** потому что нападающий использует словарь общих ключей. Дэниел Кляйн (Daniel Klein) смог расколоть 40 процентов паролей на среднем компьютере, используя этот способ вскрытия. Он не перебирал один пароль за другим, пытаясь войти в систему. Он скопировал зашифрованный файл паролей и предпринял вскрытие автономно. Вот что он пробовал:

1. В качестве возможного пароля - имя пользователя, инициалы, имя бюджета и другую связанную с человеком информацию. В целом на основе такой информации пробовалось до 130 различных паролей. Вот некоторые из паролей, проверявшихся для имени бюджета **klone** и пользователя "Daniel V. Klein":

klone, klone0, klonel, klonel23, dvk, dvkdvk, dklein, Dklein, leinad, nielk, dvklein, danielk, DvkkvD, DANIEL-KLEIN, (klone), KleinD, и так далее.

2. Слова из различных баз данных. Использовались списки мужских и женских имен (всего около 16000), названия мест (включая изменения, поэтому рассматривались и "spain", " Spanish", и "Spaniard"), имена известных людей, мультфильмы и мультипликационные герои, заголовки, герои и места из фильмов и научной фантастики, мифические существа (добытые из *Bullfinch's Mythology* и словарей мифических животных), спорт (включая названия команд, прозвища и специальные термины), числа (записанные как цифрами - '2001", так и буквами "twelve"), строки символов и чисел ("a", "aa", "aaa", "aaaa" и т.д.), китайские слоги (из Piny in Romanization of Chinese - международного стандарта письма по-китайски на англоязычной клавиатуре), Библия короля Джеймса; биологические термины, разговорные и вульгарные выражения (типа "fuckyou", "ibmsux" и "deadhead"), стандарты клавиатуры (типа "qwerty", "asdf" и "zxcvbn"), сокращения (типа "roygbiv" - первые буквы названий цветов радуги по-английски "ooottafagvah" - мнемоническая схема для запоминания 12 черепных нервов), имена компьютеров (полученные из /etc/hosts), герои пьес и места действия у Шекспира, самые распространенные слова языка Идиш, названия астероидов, совокупность слов из различных технических статей, опубликованных ранее Кляйном. Итого, для пользователя рассматривалось более чем 60000 отдельных слов (с отбрасыванием дубликатов в различных словарях).
3. Вариации слов из пункта 2. Это включало перевод первого символа в верхний регистр или его замену управляющим символом, перевод всего слова в верхний регистр, инверсию регистра слова (с и без вышеупомянутого изменения регистра первой буквы), замену буквы "o" на цифру "0" (так, чтобы сл о- во "scholar" было также проверено как "sch0lar"), замену буквы " 1" на цифру "1" (так, чтобы слово "scholar" было бы также проверено как "scho1ar") и выполнение аналогичных манипуляций с буквой "z" и цифрой "2", а также с буквой "s" и цифрой "5". Другая проверка состояла из перевода слова во множественное число (независимо от того, было ли слово существительным) с учетом необходимых правил, чтобы "dress" заменилось на "dresses", "house" - на "houses", а "daisy" - на "daisies". Хотя Кляйн не жестко придерживался правил преобразования ко множественному числу, поэтому "datum" стала "datums" (а не "data"), "sphynx" - "sphynxs" (а не "sphynges"). Аналогично, для преобразования слов добавлялись суфиксы " -ed", "-er" и "-ing", подобно "phase" в "phased," "phaser" и "phasing". Эти дополнительные проверки добавили еще 1000000 слов к списку возможных паролей, которые проверялись для каждого пользователя.
4. Различные варианты преобразования к верхнему регистру слов пункта 2, не рассматривавшихся в пункте 3. Сюда вошло преобразование к верхнему регистру одиночных символов (так, чтобы "michael" был также проверен как "m Ichael", "miChael", "micHael", "michAel", и т.д.), преобразование к верхнему регистру пары символов ("Michael", "MiChael", "MicHael", ..., "m IChael", "mIcHael", и т.д.), преобразование к верхнему регистру трех символов, и т.д. Изменения одиночного символа добавили к проверяемым примерно еще 400000 слов, а изменения пары символов - 1500000 слов. Изменения трех символов добавляли по крайней мере еще 3000000 слов для каждого пользователя, если для завершения тестирования хватало времени. Проверка изменения четырех, пяти и шести символов была признана непрактичной, так как для их проверки не хватало вычислительных мощностей.

5. Иностранные слова для иностранных пользователей. Специфический тест, который был выполнен, проверял пароли из китайского языка для пользователей с китайскими именами. Для китайских слогов использовался стандарт Pinyin Romanization, слоги объединялись вместе в одно, двух и трех сложные слова. Так как не было выполнено предварительной проверки слов на значимость, использовался исчерпывающий перебор. Так как в системе Pinyin существует 298 китайских слогов, то имеется 158404 слов с двумя слогами, и немного больше 16000000 слов с тремя слогами. Подобный способ вскрытия мог бы быть легко использован и для английского языка, с учетом правил образования произносимых ничего не значащих слов.
6. Пары слов. Объем такого исчерпывающего теста колеблется. Чтобы упростить тест, из */usr/dict/words* использовались только слова длиной три или четыре символа. Даже при этом, число пар слов составило приблизительно десять миллионов.

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

Случайные ключи

Хорошими ключами являются строки случайных битов, созданные некоторым автоматическим процессом. Если длина ключа составляет 64 бита, то все возможные 64-битовые ключи должны быть равновероятны. Генерируйте биты ключей, пользуясь либо надежным источником случайных чисел, либо криптографически безопасным генератором псевдослучайных битов.

Для генерации ключей важно использовать хороший генератор случайных чисел, но гораздо важнее использовать хорошие алгоритмы шифрования и процедуры управления ключами.

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

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

Генерация случайного ключа возможна не всегда. Иногда вам нужно помнить ваш ключ. Если вам надо генерировать простой для запоминания ключ, замаскируйте его. Идеалом является то, что легко запомнить, но трудно угадать. Вот несколько предложений:

— Пары слов, разделенные символом пунктуации, например, " turt1e*moose" или "zorch!sp1at"

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

Ключевые фразы

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

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

может "перемолоться" в такой 64-битовый ключ:

e6cl 4398 5ae9 0a9b

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

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

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

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

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

Стандарт генерации ключей X9.17

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

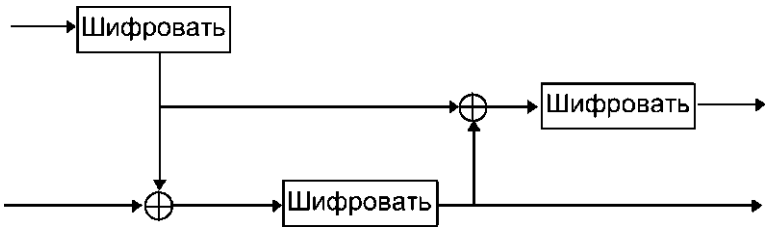


Рис. 4.1 Генерация ключей ANSI X9.17

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

$R= E_k(E_k(T_i) \oplus V_i)$

Для генерации V_i+_1, вычислим:

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

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

Нелинейные пространства ключей

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

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

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

Это работает только, если используется секретный алгоритм, который враг не может перепроектировать, или если различие в силе ключей достаточно тонко, чтобы враг не смог о нем догадаться.

Передача ключей

Алиса и Боб собираются для безопасной связи использовать симметричный криптографический алгоритм, им нужен общий ключ. Алиса генерирует ключ, используя генератор случайных ключей. Теперь она должна безопасно передать его Бобу. Если

Алиса сможет где-то встретить Боба, то она сможет передать ему копию ключа. В противном случае у них есть проблема. Криптография с открытыми ключами решает проблему легко и с минимумом предварительных соглашений, но эти методы не всегда доступны. Некоторые системы используют альтернативные каналы, считающиеся безопасными. Алиса могла бы посылать Бобу ключ с доверенным посыльным.

Алиса могла бы послать Бобу симметричный ключ по их каналу связи - тот, который они собираются шифровать. Но не правильно передавать ключ шифрования канала по этому же каналу в открытом виде, кто-то, подслушивающий канал, наверняка сможет расшифровывать все сообщения.

Стандарт X9.17 определяет два типа ключей: **ключи шифрования ключей** и **ключи данных.** Ключами шифрования ключей при распределении шифруются другие ключи. Ключи данных шифруют сами сообщения. Ключи шифрования ключей должны распределяться вручную, (хотя они могут быть в безопасности в защищенном от взлома устройстве, таком как кредитная карточка), но достаточно редко. Ключи данных распределяются гораздо чаще. Эта идея двухсвязных ключей часто используется при распределении ключей.

Другим решением проблемы распределения является разбиение ключа на несколько различных частей и передача их по различным каналам. Одна часть может быть послана телефоном, другая - почтой, третья - службой доставки, четвертая - почтовым голубем, и так далее. Так противник может собрать все части, кроме одной, и все равно ничего не узнает про ключ. Этот метод будет работать во всех случаях, кроме крайних.

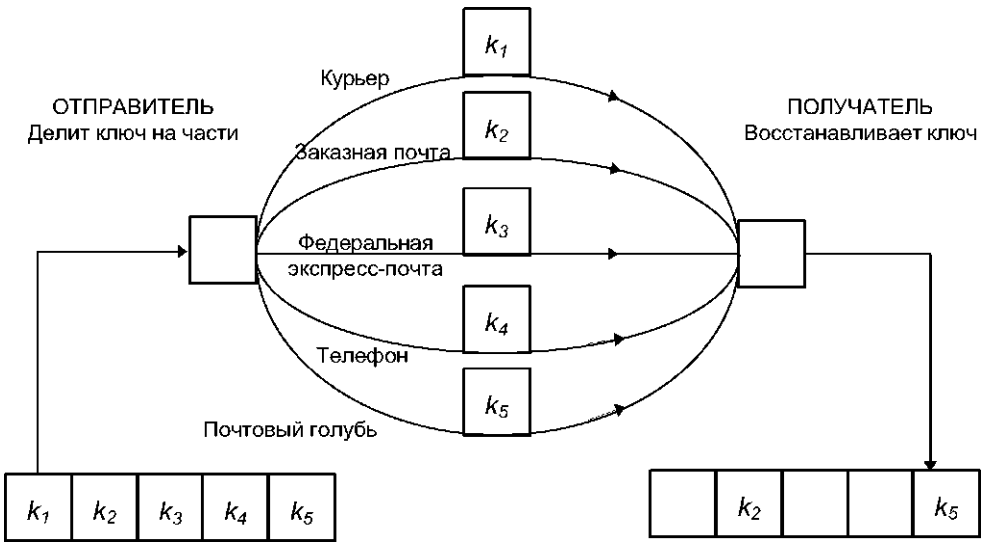


Рис. 4.2 Распределение ключей по параллельным каналам.

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

Распределение ключей в больших сетях

Ключи шифрования ключей, общие для пары пользователей, хорошо использовать в небольших сетях, но с увеличением сети такая система быстро становится громоздкой. Так как каждая пара пользователей должна обменяться ключами, общее число обменов ключами в сети из n человек равно n(n - l)/2.

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

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

Проверка ключей

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

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

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

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

Обнаружение ошибок при передаче ключей

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

Одним из наиболее широко используемых методов является шифрование ключом некоторой постоянной величины и передача первых 2 - 4 байт этого шифротекста вместе с ключом. У получателя делается то же самое. Если шифрованные константы совпадают,

то ключ был передан без ошибки. Вероятность ошибки находится в диапазоне от $1/2^{16}$ до $1/2^{32}$.

Обнаружение ошибок при дешифрировании

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

Наивным подходом явилось бы присоединение к открытому тексту до шифрования **проверочного блока** - известного заголовка. Получатель Боб расшифровывает заголовок и проверяет, что он правилен. Это работает, но дает Еве известный кусочек открытого текста, что помогает ей криптоанализировать систему. Это также облегчает вскрытие шифров с коротким ключом, таких как DES и все экспортируемые шифры. Для устранения этого недостатка рассчитайте заранее один раз для каждого ключа проверочную сумму, затем используйте эту проверочную сумму для определения ключа в любом сообщении, которое вы перехватили после этого. *Любая* проверочная сумма ключа, в которую не включены случайные или, по крайней мере, различные данные, обладает этим свойством. По идее это очень похоже на генерацию ключей по ключевым фразам.

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

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

(3) Хэшируйте результат.

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

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

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

Использование ключей

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

Аппаратные реализации безопаснее. Многие из устройств шифрования разработаны так, чтобы любое вмешательство приводило бы к уничтожению ключа. Например, в плате шифрования для IBM PS/2 залитый эпоксидной смолой модуль содержит микросхему

DES, батарею и память. Конечно, Вы должны верить, что производитель аппаратуры правильно реализовал все необходимые свойства.

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

Контроль использования ключей

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

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

Обновление ключей

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

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

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

Время жизни ключей

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

— Чем дольше используется ключ, тем больше вероятность его компрометации. Люди записывают ключи и теряют их. Происходят несчастные случаи. Если вы

используете ключ в течение года, то вероятность его компрометации гораздо выше чем, если бы вы использовали его только один день.

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

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

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

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

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

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

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

Время жизни закрытых ключей для приложений криптографии с открытыми ключами зависит от приложения. Закрытые ключи для цифровых подписей и идентификации могут использоваться годами (даже в течение человеческой жизни). Закрытые ключи для протоколов бросания монеты могут быть уничтожены сразу же после завершения протокола. Даже если считается, что время безопасности ключа примерно равно

человеческой жизни, благоразумнее менять ключ каждую пару лет. Во многих четях закрытые ключи используются только два года, затем пользователь должен получить новый закрытый ключ. Старый ключ, тем не менее, должен храниться в секрете на случай, когда пользователю будет нужно подтвердить подпись, сделанную во время действия старого ключа. Но для подписания новых документов должен использоваться новый ключ. Такая схема позволит уменьшить количество документов, которое криптоаналитик сможет использовать для вскрытия

Управление открытыми ключами

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

— Получить ключ от Боба.
— Получить его из централизованной базы данных.
— Получить его из своей личной базы данных.

Заверенные открытые ключи

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

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

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

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

К тому же, одной пары открытый ключ/закрытый ключ недостаточно. Конечно же, любая хорошая реализация криптографии с открытыми ключами должна использовать разные ключи для шифрования и для цифровых подписей. Такое разделение разрешает

различия. Это разделение учитывает различные уровни защиты, сроки действия, процедуры резервирования, и так далее. Кто-то может подписывать сообщения 2048-битовым ключом, который хранится на интеллектуальной карточке и действует двадцать лет, а кто-то может использовать для шифрования 768-битовый ключ, который хранится в компьютере и действует шесть месяцев.

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

Распределенное управление ключами

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

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

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

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

4.2 Дифференциальный и линейный криптоанализ

Дифференциальный криптоанализ

В 1990 году Эли Бихам и Ади Шамир ввели понятие **дифференциального криптоанализа**. Это был новый, ранее неизвестный метод криптоанализа. Используя этот метод, Бихам и Шамир нашли способ вскрытия DES с использованием выбранного открытого текста, который был эффективнее вскрытия грубой силой.

Мы будем рассматривать криптоанализ на основе алгоритма DES.

Дифференциальный криптоанализ работает с **парами шифротекстов,** открытые тексты которых содержат определенные отличия. Метод анализирует эволюцию этих отличий в процессе прохождения открытых текстов через этапы DES при шифровании одним и тем же ключом.

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

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

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

Определенные различия пар открытых текстов обладают высокой вероятностью вызвать определенные различия получаемых шифротекстов. Эти различия называются **характеристиками.** Характеристики распространяются на определенное количество этапов и по существу определяют прохождение этих этапов. Существуют входное различие, различие на каждом этапе и выходное различие - с определенной вероятностью. Эти характеристики можно найти, создав таблицу, строки которой представляют возможные входы XOR (XOR двух различных наборов входных битов), столбцы - возможные результаты XOR, а элементы - сколько раз конкретный результат XOR встречается для заданного входа XOR. Такую таблицу можно сгенерировать для каждого из восьми S-блоков DES.

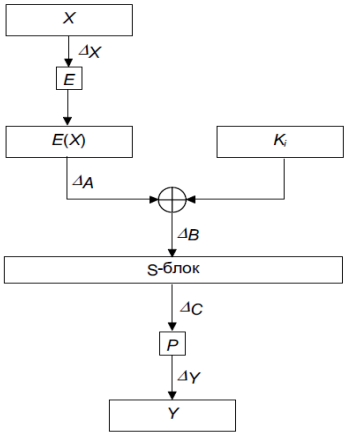


Рис. 4.3 Функция этапа DES.

Например, на рисунке 4.4 характеристика DES а) ниже показана характеристика одного этапа. Входное различие слева равно L, оно может быть произвольным. Входное различие справа равно 0. (У двух входов одинаковая правая половина, поэтому их различие - 0.) Так как на входе функции этапа нет никаких различий, то нет различий и на выходе функции этапа. Следовательно, выходное различие левой части - L XOR 0 = L, а выходное различие правой части - 0. Это тривиальная характеристика, она истинна с вероятностью 1.

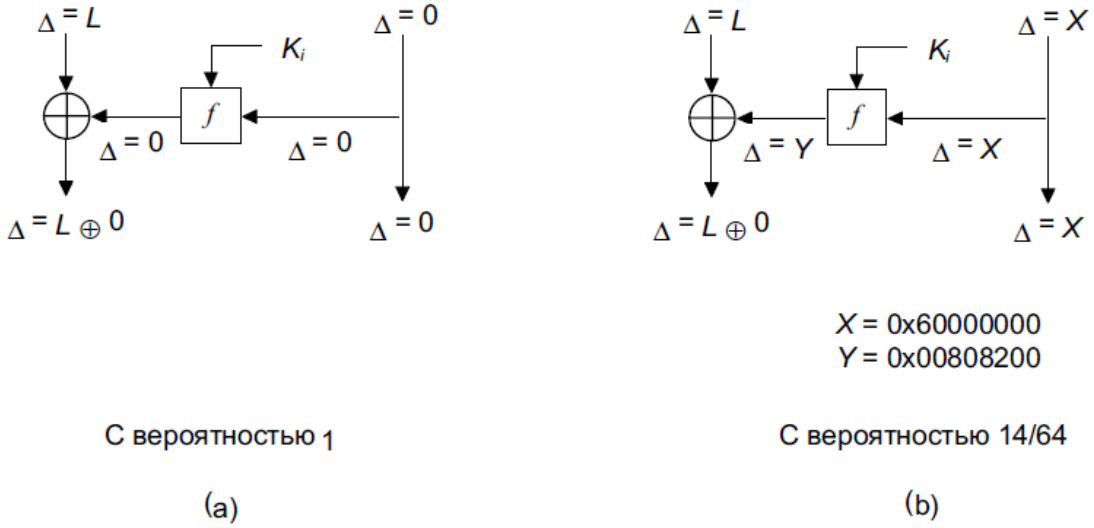
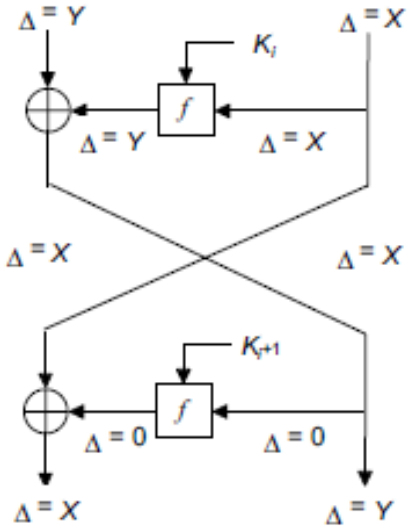


Рис. 4.4 Характеристики DES.

На рисунке б) показана менее очевидная характеристика. Снова, различие L левых частей произвольно. Входное различие правых частей равно 0x60000000, два входа отличаются только первым и третьим битами. С вероятностью 14/64 различие на выходе функции этапа равно L XOR 0x00808200. Это означает, что выходное различие левых половин равно L XOR 0x00808200, а выходное различие правых половин - 0x60000000 (с вероятностью14/64)

Различные характеристики можно объединять. Также, при условии, что этапы независимы, вероятности могут перемножаться. На рисунке 4.5 с раундом DES объединяются две ранее описанных характеристики. Входное различие слева равно 0x00808200, а справа - 0x60000000. В конце первого этапа входное различие и результат функции этапа нейтрализуют друг друга, и выходное различие равно 0. Это различие поступает на вход второго этапа, окончательное выходное различие слева равно 0x60000000, а справа - 0. Вероятность этой двухэтапной характерист ики - 14/64.



X = 0x60000000
Y = 0x00808200

С вероятностью 14/64

(b)

Рис. 4.5 Двухэтапная характеристика DES.

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

Чтобы найти правильный ключ этапа, нужно просто собрать достаточное количество предположений. Один из подключей будет встречаться чаще, чем все остальные. Фактически, правильный подключ возникнет из всех случайный возможных подключей.

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

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

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

Результаты являются весьма интересными. В таблице 4.3 проведен обзор лучших дифференциальных вскрытий DES с различным количеством этапов. Первый столбец содержит количество этапов. Элементы следующих двух столбца представляют собой количество выбранных или известных открытых текстов, которые должны быть проверены для вскрытия, а четвертый столбец содержит количество действительно проанализированных открытых текстов. В последнем столбце приведена сложность анализа, после обнаружения требуемой пары.

Табл. 4.3 Вскрытие с помощью дифференциального криптоанализа

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

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

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

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

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

Нужно отметить ряд важных моментов. Во первых, это вскрытие в значительной степени теоретическое. Огромные требования к времени и объему данных, необходимых для выполнения вскрытия с помощью дифференциального криптоанализа, находятся почти для всех вне пределов досягаемости. Чтобы получить нужные данные для

выполнения такого вскрытия полного DES, вам придется почти три года шифровать поток выбранных шифротекстов 1.5 Мегабит/с. Во вторых, это в первую очередь вскрытие с выбранным открытым текстом. Оно может быть преобразовано к вскрытию с известным открытым текстом, но вам придется просмотреть все пары "открытый текст/шифротекст" в поисках полезных. В случае полного 16-этапного DES это делает вскрытие чуть менее эффективным по сравнению с грубой силой (вскрытие дифференциальным криптоанализом требует 2^{551} операций, а вскрытие грубой силой - 2^{55}). Таким образом, правильно реализованный DES сохраняет устойчивость к дифференциальному криптоанализу.

Почему DES так устойчив к дифференциальному криптоанализу?

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

Криптоанализ со связанными ключами

В 3-х-этапном линейном приближении DES показано количество битов, на которые циклически смещается ключ DES на каждом этапе: на 2 бита на каждом этапе, кроме этапов 1, 2, 9 и 16, когда ключ сдвигается на 1 бит.

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

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

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

Линейный криптоанализ

Линейный криптоанализ представляет собой другой тип криптоаналитического вскрытия, изобретенный Мицуру Мацуи (Mitsuru Matsui). Это вскрытие использует линейные приближения для описания работы блочного шифра (в данном случае DES.)

Это означает, что если вы выполните операцию XOR над некоторыми битами открытого текста, затем над некоторыми битами шифротекста, а затем над результатами, вы получите бит, который представляет собой XOR некоторых битов ключа. Это называется линейным приближением, которое может быть верным с некоторой вероятностью p. Если $p \geq 1/2$, то это смещение можно использовать. Используйте собранные открытые тексты и связанные шифротексты для предположения о значениях

битов ключа. Чем больше у вас данных, тем вернее предположение. Чем больше смещение, тем быстрее вскрытие увенчается успехом.

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

Если линейные приближения не смещены, то они будут выполняться для 32 из 64 возможных входов. Наиболее смещенным S-блоком является пятый S-блок, для 12 входов второй входной бит равен XOR всех четырех выходных битов. Это соответствует вероятности 3/16 или смещению 5/16, что является самым большим смещением для всех S-блоков.

Было показано, как воспользоваться этим для вскрытия функции этапа DES. b_{26} - это входной бит S-блока 5. Биты нумеруются слева направо от 1 до 64. Мацуи игнорирует это принятое для DES соглашение и нумерует свои биты справа налево и от 0 до 63.

c_{17}, c_{18}, c_{19}, c_{20} - это 4 выходных бита S-блока 5. Мы можем проследить b_{26} в обратном направлении от входа в S-блок. Для получения b_{26} бит объединяется с помощью XOR с битом подключа $K_{i,26}$. А бит X_{17} проходит через подстановку с расширением, чтобы превратиться в a_{26}. После S-блока 4 выходных бита проходят через P-блок, превращаясь в четыре выходных бита функции этапа: Y_3, Y_8, Y_{14} и Y_{25}. Это означает, что с вероятностью 1/2 - 5/6:

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

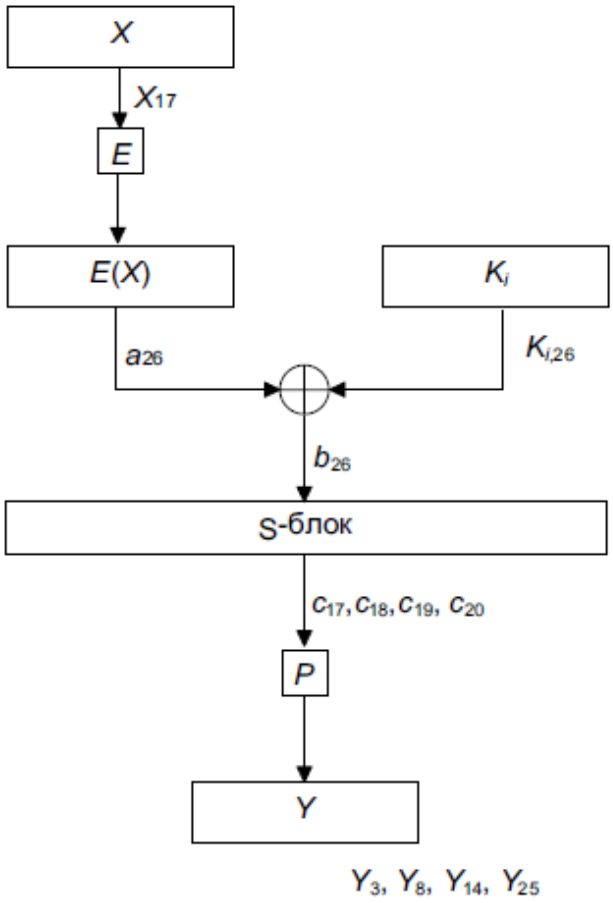


Рис. 4.6 1-этапиое линейное приближение для DES.

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

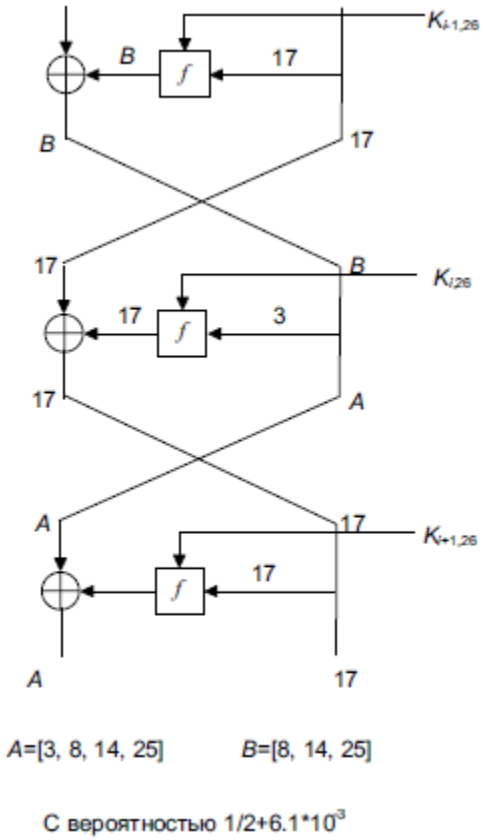


Рис. 4.7 3-этапное линейное приближение DES.

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

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

При вскрытии таким образом полного 16 этапного DES ключ будет раскрыт в среднем с помощью 2^{43} известных открытых текстов. Программная реализация этого вскрытия, работала на 12 рабочих станциях HP9735, раскрыла ключ DES за 50 дней. Линейный криптоанализ сильно зависит от структуры S-блоков, оказалось, что S-блоки DES не оптимизированы против такого способа вскрытия. Действительно, смещение в S-блоках, выбранных для DES, находится между 9 и 16 процентами, что не обеспечивает надежной защиты против линейного криптоанализа. Согласно Дону Копперсмиту устойчивость к линейному криптоанализу "не входило в число критериев проектирования DES". Либо разработчикам не было известно о линейном криптоанализе, либо при проектировании они отдали преимущество устойчивости против известного им еще более мощного средства вскрытия.

Линейный криптоанализ новее, чем дифференциальный, и в ближайшее время возможно дальнейшее продвижение в этом направлении.

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

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

4.3 Сравнение длин симметричных и открытых ключей

Длина симметричного ключа

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

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

Для выполнения такого вскрытия криптоаналитику требуется кусочек шифротекста и соответствующего открытого текста, вскрытие грубой силой представляет собой вскрытие с известным открытым текстом. Для блочного шифра криптоаналитику понадобится блок шифротекста и соответствующий открытый текст: обычно 64 -128 бит. Заполучить такие кусочки открытого текста и шифротекста легче, чем можно себе представить. Криптоаналитик может получить каким-то образом копию открытого текста сообщения и перехватить соответствующий шифротекст. Он может знать что-то о формате шифротекста: например, что это файл в формате WordPerfect, или у него есть стандартный заголовок сообщения электронной почты, или файл каталога UNIX, или изображение в формате TIFF, или стандартная запись в базе данных клиентов. Все эти форматы содержат некоторые предопределенные байты. Криптоаналитику для такого вскрытия не нужно много открытого текста.

Рассчитать сложность вскрытия грубой силой нетрудно. Если используется 8-битовый ключ, то существует 2^8, или 256, возможных ключей. Следовательно, для обнаружения правильного ключа потребуется, самое большее, 256 попыток, с 50-процентной вероятностью найти нужный ключ после половины попыток. Если длина ключа равна 56 битам, то существует 2^{56} возможных ключей. Если компьютер может проверить миллион ключей в секунду, поиск нужного ключа займет в среднем 2285 лет. Если используется 64-битовый ключ, то тому же суперкомпьютеру понадобится около 585000 лет, чтобы найти правильный ключ среди 2^{64} возможных ключей. Если длина ключа равна 128 битам поиск ключа займет 10^{25} лет. При 2048-битовом ключе миллион компьютеров, работая параллельно и проверяя миллион ключей в секунду, потратят 10^{587} лет в поисках ключа.

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

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

Оценки времени и стоимости вскрытия грубой силой

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

Скорость вскрытия грубой силой определяется двумя параметрами: количеством проверяемых ключей и скоростью проверки одного ключа. Большинство симметричных алгоритмов в качестве ключа могут использовать в качестве ключа любую битовую последовательность фиксированной длины. Длина ключа DES составляет 56 бит, всего может быть 2^{56} возможных ключей, AES от 128 бит до 256. Скорость, с которой может быть проверен каждый ключ, имеет менее важное значение. Для проводимого анализа предполагается, что скорость проверки ключа для каждого алгоритма примерно одинакова. В действительности скорость проверки одного алгоритма может быть в два, три или даже десять раз выше, чем другого. Но так как для тех длин ключей, для которых мы проводим поиск, время поиска в миллионы раз больше, чем время проверки одного ключа, небольшие отличия в скорости проверки не имеют значения.

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

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

сообщения вполне окупит стоимость аппаратуры взлома. Кроме того, стоимость многих сообщений со временем очень быстро падает.

Нейронные сети

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

Вирусы

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

Взломщик пишет и выпускает на волю компьютерный вирус. Этот вирус не переформатирует жесткий диск, не удаляет файлы, но во время простоя компьютера он работает на криптоаналитической проблемой грубого взлома. Различные исследования показали, что компьютер простаивает от 70 до 90 процентов времени, так что у вируса не буд1ет проблем с временем для решения этой задачи. Если он нетребователен и в других отношениях, то его работа даже не будет заметна.

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

Термодинамические ограничения

Одним из следствий закона второго термодинамики является то, что для представления информации необходимо некоторое количество энергии. Запись одиночного бита, изменяющая состояние системы, требует количества энергии не меньше чем k_T; где T - абсолютная температура системы и k - постоянная Больцмана.

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

Длина открытого ключа

Однонаправленной функцией является умножение двух больших простых чисел, получить произведение, перемножив числа, нетрудно, но нелегко разложить произведение на множители и получить два больших простых числа. Криптография с открытыми ключами использует эту идею для создания однонаправленной функции с люком. На самом деле, *не доказано,* что разложение на множители является тяжелой

проблемой. Насколько сегодня известно, это похоже на правду. И даже если это так, никто не может доказать, что трудные проблемы действительно трудны. Все считают, что разложение на множители является трудной задачей, но это никогда не было доказано математически.

Доминирующие сегодня алгоритмы шифрования с открытым ключом основаны на трудности разложения на множители больших чисел, которые являются произведением двух больших простых чисел. Другие алгоритмы основаны на так называемой дискретной проблеме логарифма, но пока предположим, что к ней применимы те же рассуждения. Эти алгоритмы также восприимчивы к вскрытию грубой силой, но по-разному. Взлом этих алгоритмов состоит не из перебора всех возможных ключей, а из попыток разложения больших чисел на множители.

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

Сегодня 512-битовые числа уже используются в операционных системах. Разложение их на множители и полная компрометация, таким образом, системы защиты, вполне реальны. Червь в Internet мог бы сделать это за выходные.

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

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

Табл. 4.4

Разложение на множители с помощью "квадратичного решета"

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

1993	120	500
1994	129	100

Табл. 4.5
Разложение на множители с помощью решета общего поля чисел

512	30000
768	$2*10^{8}$
1024	$3*10^{11}$
1280	$1*10^{14}$
1536	$3*10^{16}$
2048	$3*10^{20}$

Кроме того, решето общего поля чисел становится все быстрее и быстрее. Математики изобретают новые способы оптимизации - и нет причин считать, что эта тенденция оборвется. Близкий алгоритм, решето специального поля чисел, уже может разлагать на множители числа определенной специализированной формы - обычно не используемые в криптографии - гораздо быстрее, чем решето общего поля чисел может разложить на множители любые числа того же размера. Разумно предположить, что решето общего поля чисел может быть оптимизировано, чтобы достичь такой же скорости. В таблице 4.6 показано количество mips-лет, требуемое для разложения чисел различной длины при помощи решета специального поля чисел.

Табл. 4.6
Разложение на множители с помощью решета специального поля чисел

512	<200
768	100000
1024	$3*10^{7}$
1280	$3*10^{9}$
1536	$2*10^{11}$
2048	$4*10^{14}$

Табл. 4.7
Рекомендованные длины открытых ключей в (битах)

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

Не забывайте учитывать значимость ключа. Открытые ключи часто используются для длительной обеспечения безопасности важной информации. Длина 1024-битовогой ключа достаточна для подписи чего-нибудь, что будет проверено в течение недели, месяца, даже нескольких лет.

Табл. 4.8 Долгосрочный прогноз разложения на множители

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

Квантовые вычисления

В основе квантовых вычислений используется двойственная природа материи (и волна, и частица). Фотон может одновременно находиться в большом количестве состояний. Классическим примером является то, что фотон ведет себя как волна, встречая частично прозрачное зеркало. Он одновременно и отражается и проходит через зеркало подобно тому, как морская волна, ударяясь о волнолом с небольшим отверстием в нем, одновременно отразится от стены и пройдет сквозь нее. Однако, при измерении фотон ведет себя подобно частице, и только одно состояние может быть обнаружено .

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

Общепризнанно, что квантовый компьютер не противоречит фундаментальным законам квантовой механики. Одним из главных препятствий в построение квантовых компьютеров является проблема некогерентности, которая является причиной потери отчетливости волновыми огибающими и приводит к сбою компьютера. Из-за некогерентности квантовый компьютер, работающий при 1К, будет сбиваться каждую наносекунду. Кроме того, для построения квантового устройства для разложения на множители потребуется огромное количество вентилей, а это может не дать построить машину. Для проекта Шора нужно совершенное устройство для возведения в степень. Внутренние часы не используются, поэтому для разложения на множители криптографически значимых чисел могут потребоваться миллионы или, возможно, миллиарды индивидуальных вентилей. Если минимальная вероятность отказа каждого из n квантовых вентилей равна p, то среднее количество испытаний, необходимое для достижения успеха, составит $(1/(1-p)$f. Число нужных вентилей растет полиномиально с ростом длины числа (в битах), поэтому число требуемых попыток будет расти с увеличением длины используемых чисел сверхэкспоненциально.

Сравнение длин симметричных и открытых ключей

Система взламывается обычно в ее слабейшем месте. Если вы проектируете систему, которая использует и симметричную криптографию, и криптографию с открытыми

ключами, то длины ключей для криптографии каждого типа должны выбираться так, чтобы вскрыть любой из компонентов системы было одинаково трудно. Бессмысленно использовать симметричный алгоритм со 128-битовым ключом вместе с алгоритмом с открытыми ключами, использующим 386-битовый ключ. Точно так же бессмысленно использовать в одной системе симметричный алгоритм с 56-битовым ключом и алгоритм с открытыми ключами, применяющий 1024-битовый ключ.

В таблице перечислены длины модулей открытых ключей, трудность разложения которых на множители сравнима со сложностью вскрытием грубой силой сопоставленных длин популярных симметричных ключей. Из таблицы 4.9 можно сделать вывод, что если вы достаточно беспокоитесь о своей безопасности, чтобы выбрать симметричный алгоритм со 112-битовым ключом, вам следует выбрать длину модуля в вашем алгоритме с открытыми ключами порядка 1792 бит. Однако, в общем случае следует выбирать длину открытого ключа более безопасную, чем длина вашего симметричного ключа. Открытые ключи обычно используются дольше и применяются для защиты большего количества информации.

Табл. 4.9

Длины симметричных и открытых ключей с аналогичной устойчивостью к вскрытию грубой силой

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

Содержание:

Список литературы:

1. О.Н. Василенко, Теоретико-числовые алгоритмы в криптографии / О.Н. Василенко –Москва: Изд-во МЦНМО, 2003. – 326 стр.
2. Н. Смарт, Криптография/ Н. Смарт- Москва: Изд-во ТЕХНОСФЕРА, 2005. – 525 стр.
3. Б. Шнайер, Прикладная криптография. Протоколы, алгоритмы, исходные тексты на языке Си / Б. Шнайер – Москва: Изд-во Триумф, 2002. -816 стр.
4. Электронно-библиотечная система. Издательство «Лань» [Электронный ресурс] Глухов М.М., Козлитин О.А., Шапошников В.А., Шишков А.Б. и др. Задачи и упражнения по математической логике, дискретным функциям и теории алгоритмов — Лань, 2008. — Режим доступа: http://e.lanbook.com/books/element.php?pl1_cid=25&pl1_id=112
5. Электронно-библиотечная система. Издательство «Лань» [Электронный ресурс] Лавров И.А., Максимова Л.Л. Задачи по теории множеств, математической логике и теории алгоритмов – Лань, 2002. – Режим доступа: http://e.lanbook.com/books/element.php?pl1_cid=25&pl1_id=2242