Криптография*

Александр Лузгарев

26 апреля 2015 г.

Содержание

*Конспект лекций спецкурса весны 2015 г.; предварительная версия.

Источники:

1. Jonathan Katz, Yehuda Lindell, *Introduction to Modern Cryptography*, Chapman & Hall/CRC, 2008.

2. Jeffrey Hoffstein, Jill Pipher, Joseph H. Silverman, *An Introduction to Mathematical Cryptography*, second edition, UTM, Springer, 2014.

1 Введение

1.1 Что такое криптография?

1.1.1. Где используется криптография?. Везде:

- передача данных (HTTPS, GSM, WiFi, Bluetooth);
- шифрование файлов на диске (EFS, TrueCrypt);
- защита контента (DVD, BluRay);
- аутентификация пользователей;
- многое другое.

Например, для передачи данных по интернету (HTTPS) используется протокол SSL/TLS; он позволяет осуществлять общения между клиентом и сервером так, что третья сторона не может «подслушивать» и не может «подменять» сообщения. Протокол SSL/TLS состоит из двух частей: сначала клиент и сервер договариваются об общем секретном ключе (*рукопожатие*), а затем с помощью этого ключа обмениваются информацией.

Традиционно в криптографии вместо слов «клиент», «сервер», и подобных, используют персонажей с именами *Алиса* и *Боб*; для обсуждения протоколов предполагают, что Алиса и Боб должны передавать друг другу сообщения (в одну сторону или в обе) так, чтобы противник не мог подслушать их, или незаметно подделать сообщения, или совершить еще какую-нибудь атаку определенного типа.

Задача шифрования файлов на диске похожа на задачу передачи данных. Здесь Алиса хранит файлы на диске в зашифрованном виде: она сохраняет файл, а позже открывает и читает его. При этом она хочет, чтобы в промежутке между записью и чтением противник не смог прочитать этот файл или модифицировать его содержание. Таким образом, можно считать, что здесь Алиса передает сообщение самой себе (в будущее), и хочет, чтобы его не могли перехватить (подслушать) или незаметно подделать.

1.1.2. Симметричное шифрование. Одна из первых идей для обмена сообщениями называется *симметричным шифрованием*. Представим, что у Алисы и Боба имеется общий ключ k, который они хранят в секрете. Алиса хочет передать Бобу сообщение m; для этого она вычисляет значение некоторой функции $E(k, m) = c$ и передает результат c Бобу. Для расшифровки Боб вычисляет значение $D(k, c)$ и получает на выходе исходное сообщение m. При этом E называется *алгоритмом шифрования*, а D — *алгоритмом дешифрования*. Шифр называется *симметричным*, поскольку функции E и D используют один и тот же ключ k.

1.1.3. Принцип Керкгофса. Обратите внимание, что единственное, что Алиса и Боб хранят в секрете — это ключ k. В частности, множество возможных сообщений $\mathcal{M}$ считается известным (как правило, оно состоит из всех возможных предложений языка, на котором ведется переписка). Кроме того, алгоритмы E и D общедоступны. Нет никакого смысла использовать закрытые (проприетарные) алгоритмы шифрования. Это соображение называется **принципом Керкгофса** (**Kerckhoffs' principle**) и было сформулировано им еще в конце позапрошлого века. Почему?

- Гораздо проще хранить в секрете и тайно передавать друг другу сравнительно небольшой (порядка сотен бит) ключ, чем алгоритм (описание или программная реализация которого может занимать в сотни раз больше места).
- В случае, если ключ перестанет быть тайной, гораздо проще использовать тот же алгоритм с другим ключом, чем менять алгоритм.
- Для обмена сообщениями между несколькими парами сторон (скажем, в одной компании) гораздо проще использовать один алгоритм и выдавать каждой паре отдельный ключ, чем использовать разные алгоритмы.

Сегодня к этим соображениям добавляются следующие: открытые алгоритмы проверяются широким кругом специалистов и могут быть стандартизированы. Кроме того, закрытые алгоритмы подвержены reverse-engineering.

1.1.4. Сколько раз используется один ключ?. Схемы шифрования можно разделить далее на два класса:

- *Однократное использование ключа*: каждый ключ используется для зашифровки только одного сообщения. Например, при отправке электронных писем для каждого письма нужно генерировать новый, уникальный ключ.
- *Многократное использование ключа*: каждый ключ используется для зашифровки нескольких сообщений. Например, как правило, на зашифрованной файловой системе один ключ используется для зашифровки многих файлов. Такие системы, как правило, сложнее, чем системы с однократным использованием, поскольку для обеспечения безопасности приходится применять дополнительные действия.

Итак, перед обменом сообщениями Алиса и Боб должны договориться о совместном секретном ключе. Первая часть их действий, таким образом, должна завершиться тем, что у Алисы и Боба есть секретный ключ k, неизвестный окружающим, и, более того, Алиса уверена в том, что она говорит именно с Бобом, а Боб — что он говорит с Алисой. После этого Алиса и Боб могут обмениваться сообщениями по определенному протоколу и использованием полученного ключа k.

1.2 Другие применения криптографии

1.2.1. Несколько примеров. Однако, криптографические алгоритмы применимы в гораздо более широкой ситуации, чем просто обмен сообщениями:

- *Цифровая подпись*. В реальном мире мы подписываем документы одной и той же подписью. В цифровом мире это невозможно — иначе хакер может просто скопировать нашу подпись и вставить в любой документ. Поэтому цифровая подпись должна зависеть от подписываемого документа.
- *Анонимный обмен сообщениями*. Предположим, что Алиса хочет поговорить с Бобом по открытому каналу связи так, чтобы Боб не знал, с кем именно он говорит.
- *Анонимная цифровая валюта*. Я хочу потратить деньги так, чтобы продавец не знал, кто я. При этом нужно предотвратить ситуацию, в которой я копирую свои деньги и трачу их дважды у разных продавцов: однократность вступает в противоречие с анонимностью. Оказывается, существуют протоколы, благодаря которым при однократном использовании валюты личность покупателя остается анонимной, а при попытке двукратного использования личность становится известной.

1.2.2. Вычисления между несколькими сторонами. Рассмотрим еще два примера.

- *Выборы.* Нам хочется устроить голосование так, чтобы после его проведения стало известно, какая партия набрала большинство голосов, но при этом все индивидуальные голоса остались в тайне. Логичное решение: создать избирательную комиссию, которой голосующие посылают свои голоса (защищенными сообщениями), и которая подсчитывает результаты и оглашает их.

- *Частные аукционы.* Стандартная практика: в аукционе побеждает тот, кто поставил больше всего денег, но платит он столько, сколько указано во второй по величине заявке (vicary auction). Снова мы хотим выяснить, кто выиграл и сколько он должен заплатить, не разглашая остальной информации (например, ставка победителя должна остаться неизвестной. Снова можно создать аукционную комиссию, которая собирает ставки и оглашает результаты.

Последние два примера являются частными случаями secure multi-party computation: мы хотим вычислить функцию от нескольких аргументов, не разглашая этих аргументов. Предложенные выше решения состоят в том, чтобы создать центр, которому доверяют все участники (trusted authority), который получает аргументы и обещает не разглашать ничего, кроме результата вычислений. Проблемы с таким решением очевидны: вообще говоря, нет никаких причин доверять такому центру, и известны массы примеров использования такого рода центров в преступных целях (ЦИК РФ). К счастью, одна криптографическая теорема утверждает, что любое вычисление, которое можно провести с использованием такого центра, можно провести и без него. При этом голосующие общаются с друг другом специальным образом так, что в конце концов у них оказывается результат вычисления нужной функции, однако они не узнают ничего про «голоса» других голосующих.

1.2.3. Еще два примера.

- *Секретный аутсорсинг вычислений.* Алиса может послать зашифрованный поисковый запрос гуглу, а гугл запустить поисковый алгоритм на этом зашифрованном запросе, получить зашифрованный же результат и послать Алисе. При этом гугл останется в полном неведении относительно того, что же было в этом запросе.

- *Доказательство с нулевым разглашением.* Пусть Алисе известно натуральное число N, являющееся произведением двух простых чисел: $N = pq$. Боб знает только N. Тем не менее, Алиса может доказать Бобу, что она знает p и q так, что Боб ничего не узнает об этих сомножителях. Эта схема работает и в других ситуациях: если Алиса знает решение некоторой задачи, она может доказать Бобу, что знает его, не разглашая этого решения.

1.2.4. План действий. Теоретическая часть криптографических рассуждений состоит из трех частей:

1. описание модели атаки;
2. описание криптографической конструкции;
3. доказательство того, что взлом хакером описанной конструкции в рамках описанной модели будет означать решение этом хакером некоторой задачи, которая считается трудной (например, разложение числа на простые множители).

Выделим четыре наиболее часто рассматриваемых типа атаки.

1. **Ciphertext-only attack**: хакер получает доступ к одному или нескольким зашифрованным сообщений и пытается дешифровать их.
2. **Known-plaintext attack**: хакеру известна одна или несколько пар сообщений вместе с их шифрами; он пытается дешифровать какое-то другое зашифрованное сообщение.
3. **Chosen-plaintext attack**: хакер может получать результат шифрования любого сообщения и пытается дешифровать какое-то зашифрованное сообщение.
4. **Chosen-ciphertext attack**: хакер может дешифровывать любое сообщение из некоторого широкого класса и пытается дешифровать какое-то сообщение, не входящее в этот класс.

Первые два типа атаки называется **пассивными**, последние два — **активными**.

1.3 История шифрования

Приведем несколько примеров [очень плохих] шифров из истории.

1.3.1. Шифр сдвига. Первый пример — **шифр Цезаря** (описан в De Vita Caesarum, Divus Iulius, ок. 110 года нашей эры): каждая буква сдвигается на три вправо (по зацикленному алфавиту). Обратите внимание, что это шифр без ключа. Очевидная модификация называется **шифром сдвига**: давайте сдвигать не на три буквы вправо, а на k (где k — это ключ). Для латинского алфавита из 26 букв имеем $k \in \{0, 1, \ldots, 25\}$. Дешифровка, разумеется, состоит в сдвиге на k букв влево. Этот шифр не очень хорош, поскольку подвержен брутфорс-атаке: достаточно перебрать все возможные значения k и выбрать из результатов дешифровки осмысленный. Это наводит на следующую мысль: в любой схеме шифрования множество возможных ключей должно быть достаточно большим, дабы полный перебор был невозможен.

1.3.2. Подстановочный шифр. Здесь k — это таблица подстановки букв. Функция E состоит в применении подстановки к буквам сообщения m, а функция D — в применении обратной подстановки. Покажем, что шифр подстановки очень легко сломать. Заметим, что количество возможных ключей равно $26! \approx 2^{88}$. Тем не менее, помогает подсчет частот букв. Так, в стандартных текстах на английском языке буква «e» встречается чаще остальных (12,7%). Найдя наиболее часто встречающуюся букву в зашифрованном сообщении, мы с большой вероятностью узнаем, куда перестановка k переводит букву «e». Следующая по частоте буква — это «t» (9,1%), что позволит нам угадать еще одну букву; затем идет «a» (8,1%).

Остальные буквы встречаются примерно с одинаковой частотой, так что мы в тупике. Следующая идея — использовать диграммы: пары подряд идущих букв. Так, буквосочетания «he», «ar», «in», «th» встречаются чаще остальных. Это позволяет находить и другие буквы.

1.3.3. Автоматизация взлома шифра сдвига. Описанный выше метод взлома не вполне автоматизирован. Опишем другой метод взлома этого шифра, также основанный на частотном анализе. Пронумеруем буквы латинского алфавита натуральными числами от 0 до 25. Пусть p_i, $0 \leqslant i \leqslant 25$ — вероятность появления i-ой буквы в обычном английском тексте (значения p_i достаточно хорошо известны). Можно подсчитать, что $\sum_{i=0}^{25} p_i^2 \approx 0{,}065$.

Пусть теперь нам дан некоторый зашифрованный текст c, и q_i — вероятность появления i-ой буквы в этом тексте (это просто количество букв i в c, деленное на длину c). Если $k \in \{0, 1, \ldots, 25\}$ — ключ, то q_{i+k} должно примерно равняться p_i (сложение здесь понимается по модулю 26). Подсчитаем значения $I_j = \sum_{i=0}^{25} p_i \cdot q_{i+j}$ для всех $j \in \{0, \ldots, 25\}$ и выберем из них то, которое ближе к 0,065 — соответствующее значение j и будет (с хорошей вероятностью) искомым ключом.

1.3.4. Шифр Виженера. Ключом k здесь является короткое слово. Для зашифровки нужно написать это слово под сообщением m несколько раз, чтобы получилось столько же букв, сколько в m, и потом сложить соответствующие буквы по модулю 26. Дешифровка выполняется аналогично с использованием вычитания вместо сложения.

Как взломать шифр Виженера? Давайте предположим, что нам известна длина n ключевого слова k. Разобьем наше зашифрованное сообщение на блоки длины n и посмотрим на первые буквы в каждом блоке. Все они зашифрованы сдвигом на одну и ту же величину — значение первой буквы ключевого слова. Поэтому можно перебрать 26 вариантов сдвига и выбрать то, которое дает значения вероятностей, близкое к естественному (например, используя метод, описанный выше для взлома шифра сдвига). Повторяя это действие с остальными буквами, можно узнать все слово k. Отметим, что нам нужно будет перебрать $26n$ вариантов, в то время как количество возможных ключей равно 26^n.

Если же длина ключевого слова неизвестна, можно повторить описанный алгоритм для каждого возможного значения $n = 1, 2, \ldots$ до достижения успеха. Есть однако, и менее примитивные методы определения длины ключа в шифре Виженера.

1.3.5. Как узнать длину кодового слова?. Первый из них — *метод Касиски* (середина XIX века). Он основан на том, что в английском языке (как и в других естественных языках) некоторые последовательности из двух и трех букв встречаются чаще остальных. К примеру, последовательность «the» встречается в английском тексте достаточно часто. При зашифровывании она может превращаться в разные трехбуквенные последовательности, но если она встретится два раза в одной и той же относительной позиции (относительно ключевого слова k), то эти два раза превратятся в одну трехбуквенную последовательность в зашифрованном тексте. Разность между позициями, в которых эта последовательность встретится, кратна длине кодового слова n. Поэтому, обнаружив часто встречающуюся трехбуквенную последовательность, можно вычислить наибольший общий делитель попарных расстояний между позициями, в которых она встречается, и это будет хорошим кандидатом на значение n.

Второй метод определения длины ключа — *метод индекса совпадения*. Заметим, что если n — длина ключевого слова k, то символы $c_1, c_{n+1}, c_{2n+1}, \ldots$ зашифрованы при помощи сдвига на одну и ту же величину k_1. Значит, распределение частот символов в этой последовательности должно быть таким же, как в естественном тексте. Пусть q_i — частота, с которой i-ая буква встречается в этой последовательности. Тогда q_{i+k_1} должно быть очень похоже на p_i. Поэтому $\sum q_i^2$ должно быть очень похоже на 0,065. Если же m взаимно просто с n, то в последовательности $c_1, c_{m+1}, c_{2m+1}, \ldots$ все символы должны встречаться примерно с одинаковой вероятностью $1/26$. В этом случае $\sum q_i^2$ должно быть очень похоже на $1/26 \approx 0{,}038$, что достаточно сильно отличается от 0,065.

Заметим, что все вышесказанное про шифр Виженера относится только к первому типу атак: ciphertext-only. Если возможны атаки второго типа, шифр Виженера может быть взломан тривиальным образом.

1.3.6. Роторная машина. Роторная машина устроена так: представим, что нам задана фиксированная перестановка букв алфавита. С помощью этой перестановки шифруется первая буква, после чего перестановка сдвигается по циклу (то есть, домножается на длинный цикл). С помощью полученной перестановки шифруется вторая буква, и так далее. Такие машины реализовывались на практике с помощью роторов — специальных цилиндров в пишущих машинках, которые печатали нужную зашифрованную букву при нажатии клавиши, и одновременно поворачивались на одну позицию. Разумеется, этот шифр может быть взломан аналогично шифру Виженера.

Самым известным примером роторных машин была немецкая Enigma, в которой использовалось несколько роторов (от трех до пяти), и общее количество ключей в которой могло достигать 2^{36}. Взлом сообщений, зашифрованных с помощью Enigma, был важнейшим достижением британских криптографов времен второй мировой войны, и сыграл ключевую роль во многих эпизодах той войны.

1.3.7. Стандарт шифрования данных. Шифрование DES (Data Encryption Standard), разработанное в IBM (1974), стало федеральным стандартом. Оно основано на той же идее, что и шифр подстановки. Число ключей в DES равно 2^{56}. Отметим, что (в отличие от роторных машин) теперь шифруется не каждый символ отдельно, а блоки длиной 64 бита. В настоящее время DES не используется из-за маленького размера пространства ключей, вполне доступного для перебора.

Сегодня применяются стандарты AES, Salsa20, и прочие, о которых мы будем говорить далее.

2 Шифрование с закрытым ключом

2.1 Шифр Вернама и совершенная секретность

2.1.1. Определение шифра.

Определение 2.1.1.1. Пусть $\mathcal{K}$, $\mathcal{M}$, $\mathcal{C}$ — произвольные множества. **Шифром** на тройке $(\mathcal{K}, \mathcal{M}, \mathcal{C})$ называется пара функций $E\colon \mathcal{K}\times\mathcal{M} \to \mathcal{C}$, $D\colon \mathcal{K}\times\mathcal{C} \to \mathcal{M}$ таких, что $D(k, E(k, m)) = m$ для всех $k \in \mathcal{K}$, $m \in \mathcal{M}$.

Замечания 2.1.1.2.

- Мы интерпретируем множество $\mathcal{K}$ как множество всевозможных **ключей** для нашего шифра, $\mathcal{M}$ как множество сообщений, которые он может зашифровать, и $\mathcal{C}$ как множество всевозможных зашифрованных сообщеий.
- Множества $\mathcal{K}$, $\mathcal{M}$, $\mathcal{C}$ всегда предполагаются конечными.
- На самом деле всегда требуется, чтобы функции D и E были не произвольными, а *вычислимыми*; более того, алгоритм, вычисляющий D и E должен быть *эффективным* в некотором смысле. Условие эффективности можно трактовать по-разному; иногда достаточно, чтобы алгоритм был полиномиальным, а для практических приложений и этого может оказаться недостаточно, и возникают конкретные ограничения на время работы алгоритма.
- Алгоритм E может быть рандомизированным, то есть, зависеть и от некоторой случайной переменной. Алгоритм D всегда предполагается детерминистским.

2.1.2. Шифр Вернама.

Пример 2.1.2.1 (Шифр Вернама (схема одноразовых блокнотов) (Vernam, 1917))**.** Пусть $\mathcal{K} = \mathcal{M} = \mathcal{C} = \{0,1\}^n$. Таким образом, ключ в нашем шифре имеет ту же длину, что и сообщение. Положим $E(k, m) = k \oplus m$ и $D(k, c) = k \oplus c$. Очевидно, что $D(k, E(k, m)) = D(k, k \oplus m) = k \oplus k \oplus m = m$.

Заметим, что алгоритмы для вычисления E и D работают чрезвычайно быстро. В то же время, непонятно, как использовать такой шифр на практике: если Алиса хочет передать Бобу сообщение, то перед этим она должна как-то секретным образом передать ему ключ, который имеет ту же длину, что и сообщение; а если у нее есть способ это сделать, то почему бы не передать сразу сообщение этим способом? Даже если забыть про это неудобство, остается главный вопрос: а хороший ли это шифр? Мы пока вообще не сказали ничего про то, какой шифр считать хорошим, а какой плохим. Конечно, это не вполне точно сформулированный вопрос, поэтому на него можно давать разные ответы. Один из первых ответов на этот вопрос дается в рамках теории информации, и первым, кто задумался об этом, стал Клод Шеннон в своей знаменитой статье 1949 года. Идея Шеннона состоит в том, что зашифрованный текст должен содержать нулевую информацию относительно исходного текста. Конечно, именно Шеннон научился количественно измерять информацию, поэтому он и смог придумать такое определение.

2.1.3. Совершенная секретность.

Определение 2.1.3.1. Шифр (E, D) над $(\mathcal{K}, \mathcal{M}, \mathcal{C})$ обладает **совершенной секретностью**, если для любых двух сообщений $m_0, m_1 \in \mathcal{M}$ одной длины и для любого $c \in \mathcal{C}$ выполнено $P(E(k, m_0) = c) = P(E(k, m_1) = c)$, где k — равномерно распределенная случайная переменная на пространстве ключей $\mathcal{K}$ (обозначение: $k \xleftarrow{R} \mathcal{K}$).

Что это значит? Если хакер перехватил зашифрованное сообщение c, то вероятность того, что было зашифровано m_0, в точности равна вероятности того, что было зашифровано m_1. Фактически это означает, что этот шифр невозможно атаковать, если у атакующего есть лишь доступ к зашифрованному сообщению.

Существуют ли шифры, удовлетворяющие такому условию совершенной секретности? Ответ: да, например, шифр из примера 2.1.2.1.

Лемма 2.1.3.2. *Шифр Вернама из примера 2.1.2.1 обладает совершенной секретностью.*

Доказательство. Расшифруем условие из определения 2.1.3.1. Если $m \in \mathcal{M}$, $c \in \mathcal{C}$, то $P(E(k, m) = c)$ равна количеству ключей $k \in \mathcal{K}$, для которых $E(k, m) = c$, деленному на общее количество ключей $|\mathcal{K}|$. Для нашего шифра $E(k, m) = c$ означает, что $k \oplus m = c$. Но тогда $k \oplus m \oplus m = c \oplus m$, и поэтому $k = c \oplus m$. Это означает, что такой ключ k, который переводит заданное сообщение m в заданный зашифрованный текст c, ровно один. Поэтому искомая вероятность $P(E(k, m) = c)$ равна $1/|\mathcal{K}|$ для всех $m \in \mathcal{M}$ и $c \in \mathcal{C}$. Следовательно, $P(E(k, m_0) = c) = P(E(k, m_1) = c)$ для всех $c \in \mathcal{C}$, $m_0, m_1 \in \mathcal{M}$, что и требовалось доказать. □

2.1.4. Вероятностная переформулирова. Переформулируем определение совершенной секретности для чуть более общей ситуации. А именно, предположим, что на множестве возможных сообщений $\mathcal{M}$ задана некоторая вероятностная мера. Иными словами, не все

сообщения одинаково вероятны. Пусть M — случайная переменная, соответствующая этому распределению. Кроме того, будем считать, что и множество возможных ключей $\mathcal{K}$ является вероятностным пространством; пусть K — соответствующая случайная переменная. Пара (K, M) порождает случайную переменную $C = E(K, M)$; иными словами, отображение $E\colon \mathcal{K} \times \mathcal{M} \to \mathcal{C}$ порождает вероятностную меру на множестве $\mathcal{C}$. В большинстве случаев мы можем (не умаляя общности) считать, что все ключи в $\mathcal{K}$ равновероятны, то есть, распределение на $\mathcal{K}$ равномерно. Более того, нам удобно считать, что $P(M = m) > 0$ для всех $m \in \mathcal{M}$ (на самом деле, это требование тоже не ограничивает общности). Дадим определение совершенной секретности в этом контексте.

Определение 2.1.4.1. Шифр (E, D) над $(\mathcal{K}, \mathcal{M}, \mathcal{C})$ обладает **совершенной секретностью**, если для любого распределения вероятности на $\mathcal{M}$, и для любых двух сообщений $m_0, m_1 \in \mathcal{M}$ одной длины и для любого $c \in \mathcal{C}$ такого, что $P(C = c) > 0$ выполнено $P(C = c|M = m_0) = P(C = c|M = m_1)$,

2.1.5. Переформулировки совершенной секретности. У этого определения есть несколько экивалентных переформулировок.

Теорема 2.1.5.1. *Пусть (E, D) — шифр над $(\mathcal{K}, \mathcal{M}, \mathcal{C})$. Равносильны:*

1. *для любого распределения вероятности на $\mathcal{M}$, любого $m \in \mathcal{M}$, любого $c \in \mathcal{C}$ такого, что $P(C = c) > 0$ выполнено $P(M = m|C = c) = P(M = m)$.*
2. *для любого распределения вероятности на $\mathcal{M}$, любого $m \in \mathcal{M}$, любого $c \in \mathcal{C}$ такого, что $P(C = c) > 0$ выполнено $P(C = c|M = m) = P(C = c)$.*
3. *шифр (E, D) обладает совершенной секретностью.*

Доказательство. Для доказательства равносильности первых двух условий достаточно заметить, что (по теореме Байеса) $P(C = c|M = m) \cdot P(M = m) = P(M = m|C = c) \cdot P(C = c)$. Предположим теперь, что выполнено второе условие. Тогда $P(C = c|M = m_0) = P(C = c) = P(C = c|M = m_1)$, что и есть определение совершенной секретности. Обратно, для совершенно секретного шифра фиксируем некоторое распределение вероятности на $\mathcal{M}$ и выберем произвольные $m_0 \in \mathcal{M}$, $c \in \mathcal{C}$. Пусть $p = P(C = c|M = m_0)$. По предположению $P(C = c|M = m) = P(C = c|M = m_0) = p$ для всех $m \in \mathcal{M}$. Поэтому

$$P(C = c) = \sum_{m \in \mathcal{M}} P(C = c|M = m) \cdot P(M = m) = \sum p \cdot P(M = m) = p = P(C = c|M = m_0).$$

Рассуждение работает для произвольного $m_0 \in \mathcal{M}$. □

2.1.6. Эксперименты по взлому. Дадим еще одну переформулировку условия совершенной секретности. Оно будет дано в терминах эксперимента, который проводит хакер $\mathcal{A}$. Этот эксперимент будет называться $\mathrm{PrivK}^{\mathrm{eav}}$. Он может выполняться для произвольного шифра $S = (E, D)$ и для произвольного хакера $\mathcal{A}$; результат выполнения мы будем обозначать через $\mathrm{PrivK}^{\mathrm{eav}}_{\mathcal{A},S}$

Эксперимент выглядит так:

1. Хакер $\mathcal{A}$ выбирает пару сообщений $m_0, m_1 \in \mathcal{M}$.
2. Выбирается случайный ключ $k \in \mathcal{K}$ и случайный бит $b \leftarrow \{0, 1\}$. Вычисляется $c = E(k, m_b)$ и посылается хакеру $\mathcal{A}$.

3. Хакер $\mathcal{A}$ отвечает битом b'.

4. Результат эксперимента $\mathrm{PrivK}^{\mathrm{eav}}_{\mathcal{A},S}$ равен 1 (мы говорим, что эксперимент удался) если $b = b'$, и 0 в противном случае.

Коротко говоря, хакер пытается угадать значение загаданного бита b, выбранного для эксперимента. Заметим, что $\mathcal{A}$ всегда может обеспечить себе успех с вероятностью $1/2$, просто выбирая бит b' наугад. Нас интересует, может ли он добиться большего. Следующая теорема дает равносильное определение совершенного шифра в терминах описанного эксперимента.

Теорема 2.1.6.1. *Пусть $S = (E, D)$ — шифр на $(\mathcal{K}, \mathcal{M}, \mathcal{C})$. Равносильны:*

1. $P(\mathrm{PrivK}^{\mathrm{eav}}_{\mathcal{A},S} = 1) = 1/2$ *для любого хакера $\mathcal{A}$.*

2. *Шифр S обладает совершенной секретностью.*

Доказательство. Упражнение. □

2.1.7. Длина ключа в случае совершенной секретности. Таким образом, шифр Вернама уже достаточно хорош с теоретической точки зрения (но не очень-то хорош с практической). Существуют ли более практичные шифры, обладающие совершенной секретностью? Шеннон доказал следующую теорему:

Теорема 2.1.7.1 (Шеннон)**.** *Если шифр (E, D) над $(\mathcal{K}, \mathcal{M}, \mathcal{C})$ обладает совершенной секретностью, то $|\mathcal{K}| \geqslant |\mathcal{M}|$.*

Доказательство. Покажем, что если $|\mathcal{K}| < |\mathcal{M}|$, то шифр не обладает совершенной секретностью. Для разнообразия используем равносильное определение из теоремы 2.1.5.1. Рассмотрим равномерное распределение на $\mathcal{M}$, и пусть $c \in \mathbb{C}$ — зашифрованный текст, вероятность которого ненулевая. Рассмотрим множество $\mathcal{M}(c)$ всех сообщений, которые могут быть зашифрованы в c (посредством какого-то ключа из $\mathcal{K}$). Иными словами, $\mathcal{M}(c) = \{m \mid m = D(k, c)$ для некоторого $k \in \mathcal{K}\}$. Очевидно, что $|\mathcal{M}(c)| \leqslant |\mathcal{K}| < |\mathcal{M}|$. Это означает, что найдется сообщение $m' \in \mathcal{M} \backslash \mathcal{M}(c)$. Но тогда $P(M = m' \mid C = c\} = 0$ и не может совпадать с $P(M = m')$, что противоречит совершенной секретности. □

Это означает, что для передачи сообщения данной длины с помощью шифра с совершенной секретностью мы должны использовать ключ как минимум такой же длины. Для шифра Вернама в этой теореме достигается равенство, поэтому в некотором смысле это оптимальный шифр с условием совершенной секретности. Еще одна теорема Шеннона (которую мы не будем доказывать) утверждает, что любой «оптимальный» шифр с условием совершенной секретности очень похож на шифр Вернама.

Теорема 2.1.7.2 (Шеннон)**.** *Пусть (E, D) — шифр на $(\mathcal{K}, \mathcal{M}, \mathcal{C})$, где $|\mathcal{K}| = |\mathcal{M}| = |\mathcal{C}|$. Этот шифр обладает совершенной секретностью тогда и только тогда, когда*

- *ключ K равномерно распределен на $\mathcal{K}$;*
- *для любых $m \in \mathcal{M}$, $c \in \mathcal{C}$ существует единственный ключ $k \in \mathcal{K}$ такой, что $E(k, m) = c$.*

2.2 Потоковый шифр

Идея потокового шифра состоит в том, чтобы слегка модифицировать шифр Вернама, и вместо случайного ключа, каждый раз выбираемого заново, использовать *псевдо-случайный*.

2.2.1. Псевдо-случайный генератор.

Определение 2.2.1.1. Псевдо-случайным генератором называется функция $G\colon \{0,1\}^s \to \{0,1\}^n$, где $n \gg s$. Здесь $\{0,1\}^s$ называется **пространством зерен**.

Идея в том, что по короткому *зерну* $k \in \{0,1\}^s$ наша функция генерирует длинную последовательность бит $G(k)$. Конечно, в реальности на псевдо-случайные генераторы накладываются дополнительные условия, которые сложнее формализовать. Остановимся на двух из них:

1. функция G должна быть вычислима с помощью эффективного алгоритма;
2. последовательность $G(x)$ должна быть *похожей на случайную*.

Предположим, что у нас есть такой псевдо-случайный генератор. Ключом в потоковом шифре будет $k \in \{0,1\}^s$, и мы используем его в качестве зерна. Получаем длинную последовательность $G(k) \in \{0,1\}^n$. Полученную последовательность мы интерпретируем как ключ для шифра Вернама. Таким образом, $c = E(k,m) = m \oplus G(k)$. Для дешифровки достаточно взять $D(k,c) = c \oplus G(k)$.

2.2.2. Предсказуемость генератора. Почему этот шифр хорош? Нам нужно новое определение стойкости шифра, поскольку понятие совершенной секретности нам не поможет. Действительно, длина ключа в потоковом шифре меньше длины сообщения, и потому он не может обладать совершенной секретностью.

Пока что мы можем сказать, что генератор случайных чисел не должен быть *предсказуемым*: не должно существовать эффективного алгоритма, который по первым i битам последовательности $G(k)$ вычисляет остальные $n-i$ бит. Почему это важно? Предположим, что атакующий каким-то образом узнал первые i бит исходного сообщения m (например, электронные письма могут начинаться со стандартных заголовков). Тогда, путем вычисления XOR первых i бит сообщения m с первыми i битами зашифрованного сообщения c, можно узнать первые i бит последовательности $G(k)$, а после этого вычислить и остальные.

Для последовательности $x \in \{0,1\}^n$ будем обозначать через $x[i]$ ее i-й бит, и через $x[i,\dots,j]$ последовательность в $\{0,1\}^{j-i+1}$, состоящую из битов последовательности x с номерами от i до j.

Определение 2.2.2.1. Псевдо-случайный генератор $G\colon K \to \{0,1\}^n$ называется **предсказуемым в позиции** i, где $0 \leqslant i \leqslant n-1$, если существует эффективный алгоритм A такой, что

$$P(A(G(k)[1,\dots,i]) = G(k)[i+1]) \geqslant \frac{1}{2} + \varepsilon,$$

где $k \xleftarrow{R} \mathcal{K}$, а ε — некоторая положительная величина, которой нельзя пренебречь. В противном случае генератор G называется **непредсказуемым в позиции** i. Псевдо-случайный генератор называется **предсказуемым**, если существует позиция i, в которой он предсказуем, и **непредсказуемым**, если такой позиции не существует.

Пример 2.2.2.2. Предположим, что случайный генератор G обладает таким свойством: если мы возьмем XOR всех битов последовательности $G(k)$, то получится 1 (иными словами, количество бит, равных 1 в $G(k)$, нечетно). Очевидно, что такой генератор предсказуем: если нам даны все биты последовательности $G(k)$, кроме последнего, то последний бит определяется однозначно. Поэтому он удовлетворяет определению предсказуемости при $i = n-1$.

2.2.3. Линейный конгруэнтный генератор.

Пример 2.2.3.1. Приведем простой пример псевдо-случайного генератора, который, однако, ненадежен и не должен использоваться на практике: **линейный конгруэнтный генератор**. У него есть три параметра: целые числа a, b, и простое число p. Введем вспомогательную последовательность r: положим $r_1 = k$ — зерно, и зададим рекуррентным образом $r_{i+1} = a \cdot r_i + b \pmod p$. Последовательность $G(k)$ будет получаться конкатенацией некоторого фиксированного количества битов чисел $r_1, r_2, \ldots$. Идея состоит в том, что последовательность r_i обладает хорошими статистическими свойствами (при всех невырожденных значениях a и b): все остатки по модулю p встречаются в ней с одинаковой плотностью. Конечно, p не может быть степенью двойки, поэтому необходимо, как минимум, игнорировать старший бит r_i.

К примеру, функция `random()` стандартной библиотеки glibc языка C устроена так: в цикле происходит деятельность вида

```
r[i] = (r[i-3] + r[i-31]) % 2^32;
output r[i] >> 1;
```

Однако, этот генератор достаточно легко предсказуем. Мораль: никогда не используйте функцию `random()` в glibc для криптографических целей!

2.2.4. Пренебрежимо малые величины. Пренебрежимо малые величины: на практике удобно считать, что величина порядка $\varepsilon \geqslant 1/2^{30}$ *не* пренебрежимо мала: грубо говоря, если мы используем шифр для зашифровки одного гигабайта данных, то «накапливается» достаточно существенная вероятность события. С другой стороны, величину $\varepsilon \leqslant 1/2^{80}$ можно считать пренебрежимо малой: существуенная вероятность не «накопится» за время жизни ключа.

Есть и более строгое определение. Функция $\varepsilon\colon \mathbb{N} \to \mathbb{R}_{\geqslant 0}$ называется *не пренебрежимо малой*, если существует натуральное d такое, что $\varepsilon(\lambda) \geqslant \lambda^{-d}$ для бесконечного количества значений $\lambda \in \mathbb{N}$. Иными словами, функция ε убывает медленнее, чем обратная к некоторой полиномиальной. Обратно, функция ε называется *пренебрежимо малой*, если она убывает быстрее, чем $1/$любой многочлен. К примеру, функция $1/2^\lambda$ пренебрежимо мала, а $1/\lambda^{1000}$ — нет.

2.3 Примеры потоковых шифров

2.3.1. Атаки на шифр Вернама. Шифр Вернама называется также схемой одноразовых блокнотов, поскольку *двухразовые* блокноты ненадежны. Как следствие, ключи в потоковых шифрах также не должны использоваться более одного раза. Предположим, что Вы зашифровали два сообщения m_1, m_2 одним ключом k и получили зашифрованные сообщения $c_1 = m_1 \oplus G(k)$ и $c_2 = m_2 \oplus G(k)$. Если хакер подслушал c_1 и c_2, то он может вычислить $c_1 \oplus c_2 = m_1 \oplus G(k) \oplus m_2 \oplus G(k) = m_1 \oplus m_2$. Оказывается, естественные языки (английский, русский) и кодировка текста обладают достаточно большой избыточностью для того, чтобы можно было по сумме $m_1 \oplus m_2$ восстановить m_1 и m_2.

Пример 2.3.1.1 (Проект «Венона» (1943–1980))**.** Советские шифровальщики использовали шифр Вернама для передачи секретных сообщений в это время. Ключи генерировались посредством бросания игрального кубика: специальный человек сидел, бросал кубик и записывал результаты. Это достаточно унылое занятие, поэтому в какой-то момент они

стали повторно использовать сгенерированные ключи. Американская разведка, имея доступ только к зашифрованным сообщениям, смогла благодаря этому расшифровать около трех тысяч сообщений.

2.3.2. MS-PPTP. Для того, чтобы избежать дублирования ключей в шифре Вернама, разумно использовать следующую схему. Предположим, что нам нужно зашифровать несколько сообщений: $m_1, m_2, \ldots$. Можно считать, что псевдо-случайный генератор G генерирует достаточно длинную последовательность бит (на практике алгоритмы часто генерируют бесконечную последовательность). Нарежем ее на кусочки нужной длины и используем первый кусок для зашифровки m_1, второй — для зашифровки m_2, и так далее. Иными словами, последовательность сообщений $m_1, m_2, \ldots$ можно воспринимать как одно длинное сообщение $m_1||m_2||\ldots$ и зашифровывать ее с помощью $G(k)$.

Пример 2.3.2.1 (MS-PPTP (point-to-point transfer protocol))**.** В этом протоколе происходит обмен сообщениями между клиентом и сервером. Клиент посылает серверу сообщение m_1, получает от сервера ответ s_1, затем посылает следующее сообщение m_2, получает ответ s_2, и так далее. Для зашифровки сообщений клиент воспринимает все m_i как одну длинную строку и зашифровывает ее с помощью псевдо-случайного генератора G (как описано выше), то есть, происходит XOR конкатенации строк $m_1, m_2, \ldots$ и последовательности $G(k)$ для некоторого секретного ключа k. В этом пока что нет ничего плохого; проблема в том, что сервер зашифровывает свои ответы $s_1, s_2, \ldots$ так же *с тем же ключом* k, посылая XOR конкатенации строк $s_1, s_2, \ldots$ и последовательности $G(k)$. Мораль: нужно использовать разные ключи для зашифровки сообщений в одну и в другую сторону. Таким образом, общий секретный ключ k на самом деле должен быть парой ключей.

2.3.3. 802.11b WEP.

Пример 2.3.3.1 (802.11b WEP)**.** Этот протокол использовался для общения между клиентом и точкой доступа Wi-Fi. У клиента и точки доступа имеется общий секретный ключ k длины 104 бита. Если клиент хочет послать точке доступа некоторый *фрейм* (кусок сообщения) m, происходит следующее: он вычисляет некоторую контрольную сумму $CRC(m)$ и приписывает ее к m; результат зашифровывается с помощью потокового шифра с ключом IV||k. Здесь IV — строка длиной 24 бита, которая меняется для каждого следующего фрейма, увеличиваясь на единицу. После этого полученный зашифрованный текст посылается на точку доступа вместе со значением строки IV. Разработчики протокола хотели избежать переиспользования одного и того же ключа k, поэтому каждый новый фрейм зашифровывается с помощью нового ключа $IV \oplus k$. Проблема в том, что длина IV — всего 24 бита, поэтому через 2^{24} фреймов (чуть больше 16 миллионов) строка IV начнет повторяться (а ключ k долговременный, и остается таким же), и мы попадем в ситуацию двухразового блокнота. Кроме того, на некоторых Wi-Fi адаптерах после перезагрузки значение IV сбрасывается в 0, поэтому даже не нужно ждать 2^{24} фреймов.

Еще одна проблема этого протокола в том, что фактические ключи для потокового шифра почти не меняются: 1||k, 2||k, 3||k,... — у всех совпадают последние 104 бита. Оказалось, что псевдо-случайный генератор, используемый в протоколе WEP (он называется RC4), плохо приспособлен для ключей, настолько близких друг к другу. Статья FMS (2001) показывает, что уже 10^6 фреймов достаточно для того, чтобы узнать секретный ключ k. Впоследствии эту величину уменьшили до примерно 40 тысяч фреймов. Мораль: для зашифровки сетевого трафика лучше использовать новый ключ в каждой сессии (см., например, TLS).

2.3.4. Шифровка файловой системы.

Пример 2.3.4.1. Предположим, что зашифровка файлов на файловой системе происходит с помощью потокового шифра с одним и тем же ключом. Если Вы сохранили какой-то текстовый файл в зашифрованном виде, а потом отредактировали в нем, скажем, первую строку и сохранили снова, то (при использовании того же ключа) зашифрованный текст почти не изменился, и хакер, получив доступ к этим зашифрованным файлам, может определить, в каком именно месте файла произошли изменения. Мораль: для зашифровки данных на диске потоковый шифр плохо подходит.

Кроме этого, потоковые шифры не гарантируют сохранности сообщений; более того, сообщение, зашифрованное таким способом, легко модифицировать. Если хакер перехватывает зашифрованное сообщение $m \oplus G(k)$, он может модифицировать его с помощью строки p и получить $m \oplus G(k) \oplus p$. При дешифровке это сообщение превратится в $m \oplus p$. Таким образом, хакер смог произвести предсказуемое изменение зашифрованного сообщения. Это очень, опасно: например, если хакер знает, что электронное письмо m начинается с текста `From: Alex`, несложно понять, как модифицировать перехваченный зашифрованный текст $m \oplus G(k)$ так, чтобы получилось `From: Paul`: нужно сделать XOR с `Alex`$\oplus$`Paul` в определенном месте.

2.3.5. RC4.

Пример 2.3.5.1 (RC4, 1987)**.** Потоковый шифр RC4 используется, к примеру, в протоколах HTTPS и WEP. Общая схема такова: из зерна размером в 128 бит он делает ключ длиной в 2048 бит; после этого запускается цикл, который за одну итерацию генерирует один бит. Эти биты и используются для зашифровки сообщений.

У него есть несколько недостатков:

- *Второй* байт вывода не вполне случаен: известно, что вероятность того, что второй байт равен нулю, равна $2/256$ (а не $1/256$, как было бы в случае равномерного распределения). Имеется небольшое смещение от равномерного и для других начальных байт: сейчас рекомендовано просто пропускать первые 256 байт вывода и начинать с 257-го.
- После достаточно длинной работы алгоритма вероятность того, что следующая пара подряд идущих байт — это $(0, 0)$, равна $256^{-2} + 256^{-3}$ (вместо ожидаемой 256^{-2}).
- При использовании близких ключей результаты не вполне независимы (см. пример 2.3.3.1).

2.3.6. CSS.

Пример 2.3.6.1 (CSS)**.** Потоковый шифр CSS (Context Scrambling System) используется для кодирования DVD (для защиты DVD от воспроизведения в других регионах) и очень плох. Этот алгоритм реализован на аппаратном уровне и основан на Linear feedback shift register (LFSR). Это регистр, состоящий из нескольких битов. Значения некоторых из этих битов суммируются с помощью XOR, после чего все биты сдвигаются направо, последний бит отправляется на выход, а вместо него слева приписывается вычисленная сумма. Зерном для этого генератора служит начальное значение регистра. Такие регистры используются в нескольких системах:

- шифрование DVD (CSS) использует два регистра LFSR;

- шифрование голосового потока GSM (A5/1.2) использует три LFSR;
- шифрование Bluetooth (E0) — четыре LFSR.

Все эти шифры чрезвычайно плохо реализованы (на аппаратном уровне, поэтому сейчас сложно изменить ситуацию). Опишем подробнее ситуацию с CSS. Зерно CSS — это 40 бит (в то время действовали ограничения на экспорт криптографических алгоритмов из США с длиной ключа более 40 бит), что уже не очень много. Алгоритм использует два регистра LFSR длиной 17 и 25 бит. Зерно делится на две части: два байта + три байта. К первым двум байтам слева приписывается единица, и получается исходное значение первого регистра; к следующим трем байтам слева приписывается единица, и получается исходное значение второго регистра. После этого каждый регистр запускается на восемь циклов и генерирует восемь бит. Полученные два байта складываются по модулю 256 (и на самом деле к ним еще прибавляется бит переноса от предыдущего сложения), и получается один байт на выходе. Он используется для XOR с шифруемым содержимым диска.

Оказывается, описанная схема взламывается перебором 2^{16} вариантов. Для начала, видеопоток в DVD сжат при помощи кодека MPEG, и в начале каждого потока идет известный заголовок длиной в 20 байт. Если перед нами зашифрованное видео, и мы знаем первые 20 байт расшифровки — значит, мы знаем первые 20 байт того, что выдает нам описанная схема с двумя LFSR. Переберем все 2^{16} вариантов для начального состояния первого регистра LFSR. Для каждого из них прокрутим алгоритм LFSR, пока он не выдаст 20 байт. Вычтем их из известных 20 байт на выходе всей схемы — получим 20 байт, которые должны быть выданы вторым регистром LFSR. Оказывается, по данным 20 байтам достаточно легко понять, являются ли они результатом работы какого-нибудь LFSR длины в 25 бит (мы не будем уточнять, как именно), и если да — то каким было начальное состояние этого LFSR. После этого мы знаем начальные состояния обоих LFSR, поэтому можем расшифровать все видео.

Описанная схема взлома реально применяется в каждом open source DVD-проигрывателе в операционной системе Linux.

2.3.7. Salsa20. Приведем примеры более современных потоковых шифров. Эти шифры были разработаны в рамках проекта eStream (2008). Основная идея состоит в том, что псевдо-случайный генератор теперь принимает на вход не только зерно, но и *nonce* — уникальный секретный код; требуется, чтобы значение nonce не повторялось при данном зерне. Таким образом, алгоритм шифрования E является функцией трех параметров: $E(k, m, r) = m \oplus G(k, r)$; пара (k, r) не должна использоваться более одного раза.

Пример 2.3.7.1 (Salsa 20)**.** Псевдо-случайный генератор в этом примере является функцией $\{0,1\}^s \times \{0,1\}^{64} \to \{0,1\}^n$, где s — длина зерна — равна 128 или 256. Максимальное значение n равно 2^{73}. Ниже будем считать, что $s = 128$. Алгоритм $Salsa20$, получив на вход пару (k, r), выдает последовательность, полученную конкатенацией значений $H(k, r, 0)||H(k, r, 1)||H(k, r, 2)||\ldots$ некоторой функции H. Функция H устроена следующим образом: приняв на вход (k, r, i), она формирует 64 байта следующим образом:

- константа τ_0: 4 байта;
- зерно k: 16 байт;
- константа τ_1: 4 байта;
- nonce r: 8 байт;

- счетчик i: 8 байт;
- константа τ_2: 4 байта;
- зерно k: 16 байт;
- константа τ_3: 4 байта.

К полученному массиву длиной в 64 байта применяется взаимно однозначная функция h, к результату снова применяется h, и так далее — всего десять раз. Наконец, мы складываем исходные 64 байта и те, которые получены десятикратной итерацией функции h — результат и есть $H(k, r, i)$.

Функция h подобрана таким образом, что ее несложно реализовать аппаратным образом; кроме того, процессоры Intel серии x86 умеют вычислять эту функцию чрезвычайно быстро за счет поддержки набора инструкций SSE2.

2.4 Определения надежности псевдо-случайных генераторов

2.4.1. Статистические тесты. Пусть $G\colon \mathcal{K} \to \{0,1\}^n$ — псевдослучайный генератор. Мы хотим оценить «степень случайности» его вывода. Иными словами, мы хотим определить, когда значение функции G на случайном равномерно распределенном ключе $k \xleftarrow{R} \mathcal{K}$ «неотличимо» от случайно выбранной последовательности $r \xleftarrow{R} \{0,1\}^n$.

Определение 2.4.1.1. Статистическим тестом на $\{0,1\}^n$ называется алгоритм, на вход которого подается элемент $x \in \{0,1\}^n$, а на выходе — один бит; «0» или «1». При этом если $A(x) = 0$, мы говорим, что x **не выглядит случайно** по мнению теста A; а если $A(x) = 1$ — что x **выглядит случайно** по мнению теста A.

Пример 2.4.1.2. Пусть $A(x) = 1$ тогда и только тогда, когда разность между количеством единиц и количеством нулей в последовательности x по модулю не превосходит $10\sqrt{n}$.

Пример 2.4.1.3. Пусть $A(x) = 1$ тогда и только тогда, когда количество пар подряд идущих нулей в последовательности x отличается от $n/4$ не более, чем на $10\sqrt{n}$.

Пример 2.4.1.4. Пусть $A(x) = 1$ тогда и только тогда, когда максимальное количество идущих подряд нулей в последовательности x не превосходит $10\log_2(n)$.

Все три примера считают, что последовательность, состоящая из одних нулей, не является случайной. Последовательность, состоящая из одних единиц, не является случайной по мнению первых двух тестов, но является случайной по мнению третьего теста.

2.4.2. Преимущество статистического теста. Как определить, насколько хорош данный статистический тест A?

Определение 2.4.2.1. Пусть $G\colon \mathcal{K} \to \{0,1\}^n$ — псевдо-случайный генератор, A — статистический тест на $\{0,1\}^n$. Определим **преимущество** теста A по отношению к генератору G как

$$\mathrm{Adv}_{\mathrm{PRG}}(A, G) = \left|\mathrm{P}_{k \xleftarrow{R} \mathcal{K}}(A(G(k)) = 1) - \mathrm{P}_{r \xleftarrow{R} \{0,1\}^n}(A(r) = 1)\right| \in [0,1].$$

Интуитивно понятно, что значение $\mathrm{Adv}(A, G)$, близкое к 1, говорит о том, что тест A смог «отличить» вывод псевдо-случайного генератора G от истинно случайной последовательности. Если же значение $\mathrm{Adv}(A, G)$ близко к 0, это означает, что A не смог отличить вывод G от случайного. Вообще, для достаточно большого (не пренебрежимо малого) значения $\mathrm{Adv}(A, G)$ мы будем говорить, что *тест* A *сломал генератор* G *с преимуществом* $\mathrm{Adv}(A, G)$.

Пример 2.4.2.2. Пусть $A(x) = 0$; тогда $\mathrm{Adv}(A, G) = 0$ для любого псевдо-случайного генератора G.

Пример 2.4.2.3. Пусть генератор G обладает следующим свойством: первый бит последовательности $G(k)$ равен единице для [примерно] $2/3$ ключей в $\mathcal{K}$.

2.4.3. Надежный псевдо-случайный генератор. Наконец, мы можем дать определение *надежного псевдо-случайного генератора*: это тот, который не может быть сломан ни одним эффективным статистическим тестом.

Определение 2.4.3.1. Псевдо-случайный генератор $G\colon \mathcal{K} \to \{0,1\}^n$ называется **надежным**, если для любого эффективного статистического теста A величина $\mathrm{Adv}_{\mathrm{PRG}}(A, G)$ пренебрежимо мала.

К сожалению, пока нет ни одного псевдо-случайного генератора G, для которого была бы доказана его надежность. И это не удивительно: из существования такого генератора следует, что $P \neq NP$.

2.4.4. Непредсказуемость надежного генератора.

Предложение 2.4.4.1. *Надежный псевдо-случайный генератор является непредсказуемым.*

Доказательство. Покажем, что если псевдо-случайный генератор G предсказуем, то он ненадежен, то есть, его можно отличить от случайного некоторым эффективным тестом. Вспомним определение предсказуемости: есть некоторый эффективный алгоритм A, который по первым i битам последовательности $G(k)$ умеет предсказывать $(i+1)$-й с вероятностью, большей $1/2$ на некоторую существенную величину:

$$P(A(G(k)[1,\ldots,i]) = G(k)[i+1]) = \frac{1}{2} + \varepsilon,$$

где $k \xleftarrow{R} \mathcal{K}$. Построим статистический тест B, который отличает вывод генератора G от случайного. Он работает так: для последовательности $x \in \{0,1\}^n$ он сравнивает значение ее $(i+1)$-го бита с тем, которое предсказано по первым i битам. Иными словами,

$$B(x) = \begin{cases} 1, & \text{если } A(x[1,\ldots,i]) = x_{i+1}; \\ 0, & \text{иначе.} \end{cases}$$

Почему B отличает последовательность вида $G(k)$ от случайной? Пусть r — случайная последовательность, то есть, $r \xleftarrow{R} \{0,1\}^n$. Чему равно $P(B(r) = 1)$? В случайной последовательности значение $(i+1)$-й бит независимо от значения первых i; поэтому $P(B(r) = 1) = 1/2$. Если же теперь $G(k)$ — наша псевдо-случайная последовательность для $k \xleftarrow{R} \mathcal{K}$, то $P(B(G(k)) = 1) \geqslant 1/2 + \varepsilon$. Значит, $\mathrm{Adv}_{\mathrm{PRG}}(B, G) > \varepsilon$, то есть, является существенной величиной. Это и означает, что тест B сломает генератор G. □

2.4.5. Надежность непредсказуемого генератора. Оказывается, верно и обратное.

Теорема 2.4.5.1 (Andrew Yao, 1982). *Пусть $G\colon \mathcal{K} \to \{0,1\}^n$ — псевдо-случайный генератор. Если для всех $i \in \{0,1,\ldots,n-1\}$ генератор G непредсказуем в позиции i, то он надежен.*

Эта теорема означает, что если «предсказатели следующего бита» не могут отличить вывод G от случайного, то и никакой статистический тест не сможет.

Упомянем одно из удивительных следствий теоремы Yao. Пусть $G\colon \mathcal{K} \to \{0,1\}^n$ — псевдослучайный генератор. Предположим, что, зная последние $n/2$ бит последовательности $G(k)$, мы можем несложно вычислить первые $n/2$ бит. Тогда G предсказуем для некоторого $i \in \{0,\dots,n-1\}$. Действительно, нетрудно видеть, что такой G не может быть надежным, а потому предсказуем по теореме 2.4.5.1.

2.4.6. Вычислительная неразличимость.

Определение 2.4.6.1. Пусть P_1, P_2 — два распределения на $\{0,1\}^n$. Будем говорить, что P_1 и P_2 **вычислительно неразличимы** (обозначение: $P_1 \approx_P P_2$), если для любого эффективного статистического теста A модуль разности

$$|P_{x\leftarrow P_1}(A(x)=1) - P_{x\leftarrow P_2}(A(x)=1)|$$

является пренебрежимо малой величиной.

Это определение позволяет переформулировать определение надежности так: псевдослучайный генератор G является надежным тогда и только тогда, когда псевдо-случайное распределение $G(k)$, $k \xleftarrow{R} \mathcal{K}$, и равномерное распределение на $\{0,1\}^n$ вычислительно неразличимы.

2.5 Семантическая надежность

2.5.1. Определение шифрования с закрытым ключом. Сейчас мы можем показать, что при использовании надежного псевдо-случайного генератора мы получаем надежный потоковый шифр. Осталось узнать, что такое «надежный потоковый шифр».

Мы предполагаем, что атакующий получает доступ к одному зашифрованному сообщению.

Для формулировки вычислительной надежности нам понадобится асимптотический подход. А именно, мы будем считать, что схема шифрования зависит от некоторого натурального параметра n. Как правило, n — это длина ключа (в битах). Приведем определение надежности шифра в этом контексте:

Определение 2.5.1.1. Шифр называется **надежным**, если любой полиномиальный противник может сломать его лишь с незначительной вероятностью.

Здесь *полиномиальный противник* — это алгоритм, время работы которого ограничено полиномиальной функцией от n, а *незначительная вероятность* — функция от n, которая растет медленнее, чем любая обратная к полиномиальной, то есть, функция f такая, что для любого полинома p существует натуральное N такое, что $f(n) < 1/p(n)$ для всех $n > N$.

Осталось определить слова «шифр» и «сломать». Мы допускаем в качестве алгоритма шифрования *рандомизированные* алгоритмы. Это означает, что алгоритм может, помимо обычных (детерминированных) вычислений, «подбрасывать монетку»: выбирать равномерно распределенный случайный бит 0 или 1. Для разнообразия включим в определение шифра и [рандомизированный] алгоритм генерации ключа.

Определение 2.5.1.2. Схема шифрования с закрытым ключом — это тройка алгоритмов (G, E, D), удовлетворяющих следующим условиям.

1. Алгоритм генерации ключа G принимает на вход натуральное число n и выдает ключ k; его время работы полиномиально зависит от n. Мы будем всегда считать, что $|k| \geqslant n$ для любого ключа, который может быть получен при вычислении $G(n)$.

2. Алгоритм шифрования E принимает на вход ключ k и сообщение m, и выдает зашифрованный текст c; его время работы полиномиально зависит от $|k| + |m|$.

3. Алгоритм дешифрования D принимаеит на вход ключ k и зашифрованное сообщение c, и выдает текст m; его время работы полиномиально зависит от $|k| + |c|$.

4. $D(k, E(k, m)) = m$ для любого ключа k, который может быть результатом вычисления $G(n)$ (из этого условия следует, что D можно считать детерминистским алгоритмом).

2.5.2. Вычислительная стойкость. Опишем теперь эксперимент по взлому, аналогичный приведенному в разделе 2.1.6. Напомним, что эксперимент состоял в том, что противник A выдает два сообщения m_0, m_1, затем получает одно из них (случайным образом выбранное), зашифрованное с помощью случайно сгенерированного ключа, и пытается отгадать, какое из двух сообщений было зашифровано. Шифр назывался надежным, если для любого противника A вероятность того, что A угадает, равна $1/2$.

В это определение мы вносим следующие изменения:

- эксперимент теперь зависит от натурального параметра n;
- теперь мы рассматриваем не всех противников, а только полиномиальных;
- результат работы эксперимента — не константа (вероятность угадать), а функция от n;
- шифр будет называться надежным, если эта функция отличается от $1/2$ на пренебрежимо малую;
- по техническим причинам мы будем требовать, чтобы сообщения m_0, m_1 имели одинаковую длину.

Приведем формальные определения.

Определение 2.5.2.1. Пусть $S = (G, E, D)$ — шифр, $\mathcal{A}$ — хакер, n — натуральное число. Эксперимент $\mathrm{PrivK}^{\mathrm{eav}}_{\mathcal{A},S}(n)$ заключается в следующем:

1. Хакер получает число n и выдает два сообщения m_0, m_1.

2. Мы генерируем ключ k посредством вычисления $G(n)$ и выбираем случайным образом бит $b = 0$ или 1. Зашифрованное сообщение $E(k, m_b)$ вычисляется и посылается хакеру.

3. $\mathcal{A}$ выдает бит b'.

4. Результат эксперимента равен 1, если $b = b'$, и 0 в противном случае.

Определение 2.5.2.2. Шифр $S = (G, E, D)$ называется **вычислительно стойким**, если для любого полиномиального алгоритма $\mathcal{A}$ существует пренебрежимо малая функция $\varepsilon(n)$ такая, что $P(\mathrm{PrivK}^{\mathrm{eav}}_{\mathcal{A},S}(n) = 1) \leqslant 1/2 + \varepsilon(n)$. Здесь вероятность берется по всем бросаниям монеты алгоритмов $\mathcal{A}$, G, E, и по случайному биту b.

Полезно взглянуть на следующую переформулировку вычислительной стойкости. Обозначим через $\mathrm{PrivK}^{\mathrm{eav}}_{\mathcal{A},S}(n, b)$ результат эксперимента, аналогичного эксперименту $\mathrm{PrivK}^{\mathrm{eav}}_{\mathcal{A},S}(n)$, в котором бит b не определяется случайно, а подается в качестве аргумента.

Упражнение 2.5.2.3. *Покажите, что шифр $S = (G, E, D)$ вычислительно стойкий тогда и только тогда, когда для любого полиномиального алгоритма $\mathcal{A}$ существует пренебрежимо малая функция $\varepsilon(n)$ такая, что*

$$|P(\mathrm{PrivK}^{\mathrm{eav}}_{\mathcal{A},S}(n, 0) = 1) - P(\mathrm{PrivK}^{\mathrm{eav}}_{\mathcal{A},S}(n, 1) = 1)| \leqslant \varepsilon(n).$$

2.5.3. Стойкость потокового шифра. Теперь мы можем показать, что потоковый шифр, основанный на надежном псевдослучайном генераторе, обладает вычислительной стойкостью.

Теорема 2.5.3.1. *Если потоковый шифр основан на надежном псевдослучайном генераторе, то он вычислительно стоек.*

Доказательство. Пусть G — надежный псевдослучайный генератор. Пусть $\mathcal{A}$ — любой полиномиальный алгоритм, и пусть $\varepsilon(n) = P(\mathrm{PrivK}^{\mathrm{eav}}_{\mathcal{A},S}(n) = 1) - 1/2$. Покажем, что функция $\varepsilon(n)$ пренебрежимо мала. Для этого мы сконструируем статистический тест D, который будет отличать вывод генератора G от случайного с вероятностью $\varepsilon(n)$. Опишем, как действует D: он получает на вход некоторую строку w. Воспользуемся алгоритмом $\mathcal{A}$ и получим пару сообщений $m_0, m_1 \in \mathcal{M}$ той же длины, что и w. Выберем случайный бит b и положим $c = w \oplus m_b$. Дадим c хакеру $\mathcal{A}$ и получим от него бит b'. Тест D будет выводить 1 если $b' = b$, и 0 в противоположном случае. Почему он отличает вывод G от случайного распределения? Если w равномерно распределено на $\{0,1\}^{|w|}$, то $P(D(w) = 1) = 1/2$. Если же $w = G(k)$ для некоторого k, то $P(D(w) = 1) = P(D(G(k)) = 1) = P(\mathrm{PrivK}^{\mathrm{eav}}_{\mathcal{A},S}(n) = 1) = 1/2 + \varepsilon(n)$. Поэтому $P(D(w) = 1) - P(D(G(k)) = 1) = \varepsilon(n)$. По определению надежного генератора функция $\varepsilon(n)$ должна быть пренебрежимо малой. Это и означает вычислительную стойкость шифра. □

2.5.4. Шифрование нескольких сообщений. Напомним, что для шифрования нескольких сообщений мы обязаны каждый раз использовать новый ключ. Поясним, почему это так. Можно дать определение надежности схемы шифрования для нескольких сообщений. Представьте, что в определении эксперимента 2.5.2.1 вместо сообщений m_0 и m_1 хакер предоставляет нам два набора сообщений $(m_0^{(1)}, \ldots, m_0^{(t)})$ и $(m_1^{(1)}, \ldots, m_1^{(t)})$. Далее, как и раньше, мы выбираем случайный бит b, зашифровываем все сообщения из набора $\{m_b^{(i)}\}$ с помощью *одного и того же* ключа k и посылаем полученный набор $\{c^{(i)} = E(k, m_b^{(i)})\}$ хакеру. Хакер выдает бит b', и эксперимент удачен, если $b = b'$.

Оказывается, схема шифрования с псевдо-случайным генератором, про которую мы только что доказали, что она является вычислительно стойкой (теорема 2.5.3.1), перестает быть стойкой при шифровании нескольких сообщений. Опишем хакера, который угадывает случайный бит b с вероятностью 1. Достаточно взять наборы сообщений $(m_0^{(1)}, m_0^{(2)}) = (0 \ldots 0, 0 \ldots 0)$ и $(m_1^{(1)}, m_1^{(2)}) = (0 \ldots 0, 1 \ldots 1)$. Получив зашифровнную пару $(c^{(1)}, c^{(2)})$, можно восстановить b: если $c^{(1)} = c^{(2)}$, то $b = 0$; иначе $b = 1$.

На самом деле, это рассуждение применимо к любой схеме шифрования, в которой алгоритм шифрования E является детерминистским (зависит только от ключа k и сообщения m). Для решения этой проблемы существует два основных способа.

1. *Синхронизированный режим* был описан в разделе 2.3.2: каждый раз для шифрования мы берем следующий кусок длинного вывода псевдо-случайного генератора G. Для этого обе стороны должны помнить, на каком месте они остановились, чтобы осуществлять шифрование и дешифрование с одинаковыми кусками вывода G.

2. *Несинхронизированный режим* фактически был описан в примере 2.3.3.1. В этом режиме у псевдо-случайного генератора G на вход кроме зерна s подается еще и *начальный вектор* IV. Этот вектор будет посылаться открытым образом вместе с зашифрованным текстом. Иными словами, $E(k, m) = (IV, G(k, IV) \oplus m)$ для случайно выбранного IV. Конечно, для стойкости такого шифра мы должны наложить сильные условия на G: неформально говоря, строка $G(s, IV)$ должна быть псевдо-случайной, даже если вектор IV известен.

2.5.5. Chosen plaintext-атаки. Пока что мы рассматривали только один тип атаки: подслушивание. Предположим теперь, что наш хакер может производить chosen-plaintext-атаку, то есть, имеет доступ к оракулу, который производит шифровку сообщений, которые ему поступают, с помощью той же схемы шифрования и того же ключа, который хакер хочет взломать. Опишем формально соответствующий эксперимент

Определение 2.5.5.1. Пусть $S = (G, E, D)$ — шифр, $\mathcal{A}$ — хакер, n — натуральное число. Эксперимент $\mathrm{PrivK}^{cp}_{\mathcal{A},S}(n)$ заключается в следующем:

1. Мы генерируем ключ k посредством вычисления $G(n)$.

2. Хакер получает число n и доступ к оракулу, вычисляющему функцию $E(k, -)$. Хакер выдает два сообщения m_0, m_1.

3. Мы выбираем случайным образом бит $b = 0$ или 1. Зашифрованное сообщение $c = E(k, m_b)$ вычисляется и посылается хакеру.

4. $\mathcal{A}$ продолжает иметь доступ к оракулу $E(k, -)$ и выдает бит b'.

5. Результат эксперимента равен 1, если $b = b'$, и 0 в противном случае.

Определение *[вычислительной] стойкости относительно chosen plaintext-атак* теперь совершенно аналогично определению 2.5.2.2.

На первый взгляд, стойкость относительно chosen plaintext-атак недостижима. Действительно, хакер $\mathcal{A}$ имеет доступ к оракулу и может попросить его зашифровать сообщения m_0, m_1, и получить таким образом c_0 и c_1. Получив c, он сравнивает c с c_0 и c_1; если $c = c_0$, то $b = 0$, а если $c = c_1$, то $b = 1$. Описанная атака, однако, возможна только если алгоритм шифрования E является детерминистским. Таким образом, для достижения стойкости относительно chosen plaintext-атак мы обязаны использовать рандомизированные алгоритмы.

2.5.6. Псевдослучайные функции. Наша ближайшая цель — модифицировать потоковый шифр, основанный на псевдослучайном генераторе, чтобы получить систему шифрования, устойчивую относительно chosen plaintext-атак. Для этого нам понадобится понятие псевдослучайной функции. Напомним, что бессмысленно говорить о том, что данная конкретная строка символов псевдослучайна: можно говорить только о псевдослучайности *распределения* строк. Аналогично, одна функция $f\colon \{0,1\}^n \to \{0,1\}^n$ не может быть псевдослучайной, но псевдослучайным может быть распределение функций. Для этого удобно рассматривать *функции с ключом.*

Определение 2.5.6.1. Отображение $F\colon \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ называется **функцией с ключом**. Здесь первый аргумент F воспринимается как ключ $k \in \{0,1\}^*$. Для каждого конкретного $k \in \{0,1\}^*$ мы получаем функцию $F_k\colon \{0,1\}^* \to \{0,1\}^*$, где $F_k(x) = F(k,x)$.

Мы всегда будем предполагать, что наша функция с ключом **сохраняет длину**, то есть, что образ множества $\{0,1\}^n \times \{0,1\}^n$ лежит в $\{0,1\}^n$. Более того, мы будем считать, что для фиксированного ключа k длины n функция F_k действует из $\{0,1\}^n$ в $\{0,1\}^n$. Иными словами, $|F_k(x)| = |x| = |k|$. Это ограничение мы накладываем исключительно для удобства. Функция с ключом F называется **эффективной**, если существует полиномиальный детерминистский алгоритм, вычисляющий $F(k,x)$ по k и x.

Каждая функция с ключом F порождает естественным образом распределение на множестве всех функций $\{0,1\}^n \to \{0,1\}^n$. А именно, взятие случайного ключа $k \in \{0,1\}^n$ (с равномерным распределением) дает нам случайную функцию F_k. Функция с ключом F будет называться *псевдослучайной*, если полученное распределение функций F_k «неотличимо» от равномерного распределения на множестве всех функций из $\{0,1\}^n$ в $\{0,1\}^n$, то есть, ни один полиномиальный алгоритм не может отличить функцию вида F_k от случайно выбранной с вероятностью, существенно отличной от $1/2$.

Обозначим через Func_n множество всех отображений из $\{0,1\}^n$ в $\{0,1\}^n$. Нетрудно понять, что $|\,\mathrm{Func}_n\,| = (2^n)^{(2^n)} = 2^{n\cdot 2^n}$. Можно смотреть на это так: функция $f\colon \{0,1\}^n \to \{0,1\}^n$ однозначно задается своей *таблицей значений*. В этой таблице 2^n строк, и в строке с номером x записано n бит — значение $f(x)$. Понятно, что случайный выбор $f \in \mathrm{Func}_n$ с равномерным распределением на Func_n можно задать так: выбрать каждую строчку в таблице значений равномерно и независимо друг от друга из $\{0,1\}^n$ Иными словами, для такой случайной функции f значения $f(x)$ и $f(y)$ (при $x \neq y$) независимы друг от друга и случайно распределены на $\{0,1\}^n$.

Итак, функция с ключом F порождает 2^n возможных функций F_k, что гораздо меньше, чем общее количество возможных функций в Func_n, которое равно $2^{n\cdot 2^n}$. Тем не менее, мы хотим, чтобы для любого полиномиального алгоритма «поведение» этих наборов функций было одинаковым.

Наивная попытка формализации этого определения такова: нам хочется, чтобы любой полиномиальный алгоритм, получив на вход функцию F_k, выдавал ответ «1» примерно с той же вероятностью, что и для случайной функции. Проблема этого определения в том, что полное описание функции F_k имеет экспоненциальную длину по n (а именно, $n \cdot 2^n$). Поэтому алгоритм D не успеет за полиномиальное время изучить всю таблицу значений функции на входе.

Настоящее определение состоит в том, что алгоритму D предоставляется доступ к «оракулу» для данной функции (F_k или f). А именно, D может в любой момент спросить у оракула, чему равно значение его функции в точке x. Алгоритм D с доступом к оракулу на функции f мы будем обозначать через D^f.

Определение 2.5.6.2. Пусть $F\colon \{0,1\}^* \times \{0,1\}^* \times \{0,1\}^*$ — эффективная функция с ключом, сохраняющая длину. Она называется **псевдослучайной функцией**, если для любого вероятностного полиномиального алгоритма D (как всегда, выводящего один бит) существует пренебрежимо малая функция $\varepsilon(n)$ такая, что

$$|P(D^{F_k}(n) = 1) - P(D^f(n) = 1)| \leqslant \varepsilon(n),$$

где k выбирается равномерно случайно из $\{0,1\}^n$, а f выбирается равномерно случайно из Func_n.

Заметим, что D не знает ключа k (иначе он тривиальным образом мог бы отличить F_k от f).

Сразу отметим, что мы не знаем, существуют ли в природе псевдослучайные функции. Известно, что этот вопрос *равносилен* вопросу о существовании [надежных] псевдослучайных генераторов.

2.5.7. Шифрование с помощью псевдослучайной функции. Опишем теперь, как по псевдослучайной функции построить схему шифрования (она окажется устойчивой относительно chosen plaintext-атак).

Определение 2.5.7.1. Пусть F — псевдослучайная функция. Опишем шифр $S = (G, E, D)$ для сообщений длины n.

- функция $G(n)$ выбирает случайно равномерно распределенный ключ $k \in \{0,1\}^n$;
- функция $E(n)$, получив ключ $k \in \{0,1\}^n$ и сообщение $m \in \{0,1\}^n$, выбирает случайный равномерно распределенный вектор $r \in \{0,1\}^n$ и выводит зашифрованное сообщение $c = (r, F_k(r) \oplus m)$.
- функция $D(n)$, получив ключ $k \in \{0,1\}^n$ и зашифрованное сообщение $c = (r, s)$, выводит незашифрованное сообщение $m = F_k(r) \oplus s$.

Неформально говоря, устойчивость описанной схемы основана на том, что $F_k(r)$ выглядит как случайная строка даже для известного r; поэтому хакер, получив пару (r, s), имеет дело с чем-то очень похожим на схему одноразовых блокнотов. Проблема может возникнуть только если строка r уже была использована для шифрования какого-нибудь сообщения — например, при ответе оракула на вопрос хакера. Однако, вероятность такого события пренебрежимо мала.

Теорема 2.5.7.2. *Если F — псевдослучайная функция, то шифр, описанный в определении 2.5.7.1, вычислительно устойчив относительно chosen plaintext-атак.*

Доказательство. Проанализируем сначала устойчивость схемы, в которой вместо псевдослучайной функции F_k используется настоящая случайная функция f. Пусть $\widetilde{S} = (\widetilde{G}, \widetilde{E}, \widetilde{D})$ — схема шифрования как в определении 2.5.7.1 с тем отличием, что вместо F_k используется случайная функция f. А именно, функция $G(n)$ выбирает ключ — равномерно распределенную случайную функцию f из Func_n, а $\widetilde{E}$ и $\widetilde{E}$ работают аналогичным образом с заменой F_k на f (разумеется, эта схема не полиномиальна: алгоритм G должен вывести таблицу значений функции f, то есть, $n \cdot 2^n$ бит; но это совершенно не важно).

Мы утверждаем, что для любого хакера $\mathcal{A}$, который делает не более $q(n)$ запросов к оракулу шифрования с ключом f, выполнено неравенство

$$P(\mathrm{PrivK}^{\mathrm{cp}}_{\mathcal{A},\widetilde{S}}(n) = 1) \leqslant \frac{1}{2} + \frac{q(n)}{2^n}.$$

Покажем это. При шифровании случайно выбранного из m_0, m_1 сообщений мы генерируем случайную строку r_c и выдаем зашифрованное сообщение $c = (r_c, f(r_c) \oplus m_b)$. Рассмотрим два случая:

1. значение r_c использовалось оракулом для ответа хотя бы на один вопрос хакера $\mathcal{A}$. Вероятность этого события не превосходит $q(n)/2^n$.

2. значение r_c ни разу не использовалось оракулом для ответов на вопросы $\mathcal{A}$. Тогда значение $f(r_c)$ выглядит для $\mathcal{A}$ как случайная строка, независимая от $f(r)$ для $r \neq r_c$; поэтому вероятность того, что $\mathcal{A}$ выдаст бит b', совпадающий с b, равна в точности $1/2$ (как в схеме одноразовых блокнотов).

Несложное вычисление с условной вероятностью показывает, что нужное неравенство выполнено.

Определим теперь для фиксированного полиномиального хакера $\mathcal{A}$ функцию

$$\varepsilon(n) = P(\mathrm{PrivK}^{cp}_{\mathcal{A},S}(n) = 1) - \frac{1}{2}.$$

Применяя к $\mathcal{A}$ доказанное выше неравенство, получаем, что

$$P(\mathrm{PrivK}^{cp}_{\mathcal{A},\tilde{S}}(n) = 1) \leqslant \frac{1}{2} + q(n)/2^n$$

для некоторого многочлена q. Покажем, что если $\mathcal{A}$ может сломать наш шифр S, то существует полиномиальный алгоритм D, отличающий функции вида F_k от случайных. Этот алгоритм D получает на вход число n и доступ к оракулу $\mathcal{O}$, вычисляющему некоторую функцию $F\colon \{0,1\}^n \to \{0,1\}^n$, и хочет определить, имеет ли она вид F_k для случайно выбранного k, или является случайно выбранной из Func_n. В его распоряжении также есть полиномиальный хакер $\mathcal{A}$, который умеет ломать шифр S. Тогда D может запустить этого хакера и, каждый раз, когда тот просит оракула зашифровать сообщение m, алгоритм D будет отвечать ему следующим образом:

1. выбирать равномерно распределенную случайную строку r и $\{0,1\}^n$;
2. спрашивать значение $\mathcal{O}(r)$ у своего оракула $\mathcal{O}$ и получать ответ s;
3. возвращать хакеру $\mathcal{A}$ в качестве зашифрованного сообщения пару $(r, s \oplus m)$.

Хакер в итоге должен выдать нам два сообщения $m_0, m_1 \in \{0,1\}^n$. После этого D выбирает случайным образом бит $b \in \{0,1\}$ и

1. выбирает равномерно распределенную строку r длины n;
2. спрашивает значение $\mathcal{O}(r)$ у своего оракула $\mathcal{O}$;
3. отдает хакеру $\mathcal{A}$ тестовое зашифрованное сообщение $(r, s \oplus m_b)$.

Хакер $\mathcal{A}$ после этого начинает вычисления; в тот момент, когда он обращается к оракулу зашифровки, мы отвечаем ему так же, как описано выше. В конце концов, $\mathcal{A}$ выдает бит b'. Алгоритм D в этот момент завершает работу, выводя 1, если $b' = b$, и 0 в противном случае.

Если оракул $\mathcal{O}$, который дан алгоритму D, вычисляет равномерно распределенную случайную функцию, то с точки зрения хакера $\mathcal{A}$ он находится внутри эксперимента $\mathrm{PrivK}^{cp}_{\mathcal{A},\tilde{S}}(n)$. Если же оракул $\mathcal{O}$ вычисляет псевдослучайную функцию F_k, то с точки зрения хакера $\mathcal{A}$ происходит эксперимент $\mathrm{PrivK}^{cp}_{\mathcal{A},S}(n)$.

Таким образом,

$$P(D^{F_k}(n) = 1) - P(D^{f}(n) = 1) \geqslant \varepsilon(n) - q(n)/2^n.$$

Мы предположили, что функция F псевдослучайна, поэтому функция $\varepsilon(n) - q(n)/2^n$ должна быть пренебрежимо малой. Поэтому и $\varepsilon(n)$ пренебрежимо мала. Но это и означает, что схема S вычислительно устойчива. $\square$

Более того, описанная схема устойчива и для шифрования нескольких сообщений. Если даны сообщения $m_1, m_2 \ldots, m_l$, для их зашифровки достаточно выбрать случайные векторы $r_1, \ldots, r_l$ и вычислить $(r_1, F_k(r_1) \oplus m_1, r_2, F_k(r_2) \oplus m_2, \ldots, r_l, F_k(r_l) \oplus m_l)$. Вообще, если схема шифрования одного сообщения устойчива относительно chosen plaintext-атак, то и итеративное шифрование нескольких сообщений устойчиво относительно chosen plaintext-атак. Однако, у приведенной схемы имеется недостаток: длина зашифрованного текста в два раза больше длины исходного текста (каждый блок длины n нужно шифровать с помощью случайной строки r_i длины n, которая потом включается в шифр). Следующий класс шифров — *блочные шифры* — избавит нас от этой неэффективности.

2.5.8. Псевдослучайные перестановки. Пусть $F\colon \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ — эффективная функция с ключом, сохраняющая длину. Она называется **перестановкой с ключом**, если при каждом фиксированном k отображение $F_k(\cdot)$ биективно, то есть, является перестановкой множества $\{0,1\}^n$. Перестановка с ключом называется **эффективной**, если существует полиномиальный алгоритм, вычисляющий $F_k(x)$ по k и x, *а также* полиномиальный алгоритм, вычисляющий обратную перестановку $F_k^{-1}(x)$ по k и x. Мы снова предполагаем, что для ключа длины n мы получаем перестановку на множестве строк той же длины n; на практике часто такое требование не выполняется, но на суть дела это не влияет.

Эффективная перестановка с ключом F называется **псевдослучайной**, если она вычислительно неотличима от случайно выбранной *перестановки* в том же смысле, как и в определении 2.5.6.2 псевдослучайной функции. На самом деле, можно потребовать, чтобы она была вычислительно неотличима от случайно выбранной *функции*, и получится равносильное определение, поскольку случайно выбранная перестановка неотличима от случайно выбранной функции. Дело в том, что для доказательства того, что данная функция f *не является* перестановкой, нужно предъявить два элемента x, y такие, что $f(x) = f(y)$. Однако, вероятность найти такие элементы за полиномиальное время пренебрежимо мала. Иными словами, (упражнение!) любая псевдослучайная перестановка является и псевдослучайной функцией.

Эффективная перестановка с ключом называется **сильной псевдослучайной перестановкой**, если она вычислительно неотличима от [равномерно распределенной] случайной перестановки для алгоритмов, которым предоставляется доступ не только к оракулу, вычисляющему f, но и к оракулу, вычисляющему обратную перестановку f^{-1}. Часто на практике используются схемы, надежность которого основана на этом, более сильном требовании к перестановке с ключом; однако, мы не будем их рассматривать.

Блочные шифры действуют следующим образом: сообщение m разбивается на части $m_1, \ldots, m_l$ одинаковой длины n, и дальше каким-то образом (с использованием псевдослучайной перестановки F) из них получается зашифрованный текст c. Обсудим несколько простых способов работы блочного шифра.

1. **ECB (Electronic Code Book)**. Это самый наивный способ: каждое сообщение m_i напрямую подставляется в нашу псевдослучайную перестановку с ключом k. То есть, $c = (F_k(m_1), \ldots, F_k(m_l))$. Дешифровка происходит аналогично с использованием обратной перестановки F_k^{-1}. Как видим, здесь алгоритм шифрования детерминистский, поэтому в принципе не может быть устойчивым относительно chosen plaintext-атак. Он неустойчив и в обычном смысле (в присутствии подслушивающего хакера), поскольку пара одинаковых блоков $m_1 = m_2$ даст на выходе пару (c_1, c_1), а пара неодинаковых блоков — пару неодинаковых блоков.

2. **CBC (Cipher Block Chaining)**. Для шифрования выбирается случайный начальный вектор IV; положим $c_0 = IV$ и $c_i = F_k(c_{i-1} \oplus m_i)$ для $i = 1, \ldots, l$. На выходе получаем зашифрованное сообщение $(c_0, c_1, \ldots, c_l)$. Дешифровка происходит очевидным образом (с использованием открыто посылаемого начального вектора $c_0 = IV$). Доказано, что такая схема шифрования устойчива относительно chosen plaintext-атак. Ее недостаток, впрочем, состоит в том, что мы обязаны зашифровывать блоки последовательно один за другим; такое вычисление сложно распараллелить. Обратите внимание, что IV должен выбираться *случайно*. Может показаться, что достаточно использовать разные IV для разных сообщений: например, взять $IV = 1$ и после этого увеличивать IV для каждого нового сообщения m. Однако, такая схема не является надежной.

3. **OFB (Output Feedback)**. В этой схеме псевдослучайная перестановка используется для того, чтобы сгенерировать псевдослучайный поток, который затем складывается (посредством $\oplus$) с сообщением. А именно, сначала выбирается IV случайным образом из $\{0,1\}^n$, а затем генерируется последовательность $r_0 = IV$, $r_1 = F_k(r_0)$, $\ldots$, $r_i = F_k(r_{i-1}), \ldots$. После этого вычисляются зашифрованные блоки $c_i = m_i \oplus r_i$, и итоговый зашифрованный текст $c = (IV, c_1, \ldots, c_l)$. Эта схема также устойчива относительно chosen plaintext-атак. Здесь шифрование блоков также должно происходить последовательно, но вычисление псевдослучайного потока $r_0, r_1, \ldots$ может быть выполнено независимо от сообщения m, так что при желании можно сделать это заранее.

4. **CTR (Counter)**. Как и в случае OFB, генерируется псевдослучайный поток. Выбирается случайный вектор ctr и вычисляется последовательность $r_i = F_k(ctr+i)$ (здесь сложение происходит по модулю 2^n). После этого шифруются блоки: $c_i = r_i \oplus m_i$, а вектор ctr снова посылается в открытом виде. Оказывается, эта схема устойчива относительно chosen plaintext-атак; кроме того, вычисления здесь можно полностью распараллелить, и, как и в режиме OFB, вычислить псевдослучайный поток заранее. Кроме того, возможно расшифровать произвольный блок, не расшифровывая остальных.

2.5.9. Конструкция псевдослучайных перестановок. Мы хотим построить псевдослучайную перестановку с ключом F: $\{0,1\}^n \times \{0,1\}^l \to \{0,1\}^l$. Это означает, что для каждого k функция $F_k(x) = F(k,x)$ биективна (то есть, является перестановкой), и функции F_k, F_k^{-1} эффективно вычислимы. Мы будем называть n *длиной ключа*, а l — *длиной блока* функции F; мы формально не требуем равенства $l = n$. Для каждой конкретной конструкции параметры n, l — фиксированные константы, и потому вместо асимптотической надежности мы будем говорить о *конкретной надежности*.

Заметим, что для задания случайной перестановки на $\{0,1\}^l$ нам нужно $\ln((2^n)!) \approx n\ln(n)$ бит.

Шеннон предложил два приема построения перестановок, похожих на случайные. Для того, чтобы построить перестановку с большой длиной блока l, мы сначала выберем несколько случайных (или похожих на случайные) перестановок f_i с меньшей длиной блока. Пусть, например, мы хотим построить перестановку F с длиной блока в 128 бит. Определим ее так: ключ для F будет задавать 16 *случайных* перестановок $f_1, \ldots, f_{16}$ с длиной блока 8. Получив аргумент, $x \in \{0,1\}^{128}$, мы прежде всего разобьем его на 16 блоков по 8 бит: $x = x_1 || \ldots || x_{16}$. Положим $F_k(x) = f_1(x_1) || \ldots || f_{16}(x_{16})$. Этот прием называется *confusion*.

Конечно, полученная перестановка F не является псевдослучайной. Второй прием называется *diffusion*: биты аргумента перетасовываются некоторым заданным образом. Комбинация этих двух приемов затем повторяется несколько раз (возможно, с разными наборами $\{f_i\}$ и с разными перетасовками).

Прямое применение указанных приемов приводит к тому, что называется **substitution-permutation сеть**. При этом функции $\{f_i\}$ считаются *фиксированными*, то есть, не зависят от ключа k. Функции $\{f_i\}$ в этом контексте называются S-ящиками, поскольку они отвечают за substitution. Промежуточные перетасовки бит также считаются фиксированными. Зависимость от ключа при этом проявляется путем взятия побитовой суммы в каждом раунде. Таким образом, один раунд (с номером j) применения substitution-permutation сети состоит из следующих шагов:

1. аргумент побитово складывается с k_j;
2. результат разбивается на части, и к i-й части применяется функция f_i;
3. биты результата перетасовываются определенным образом.

Этот набор шагов повторяется R раз, то есть, выполняются раунды $j = 1, 2, \ldots, R$. Ключ для substitution-permutation сети состоит из под-ключей $k_1, k_2, \ldots, k_R$.

Полученная псевдослучайная перестановка может быть надежной (неотличимой от случайной) или нет в зависимости от выбора S-ящиков, промежуточных перетасовок и количества раундов. Приведем несколько общих принципов, которым должны удовлетворять эти параметры.

- S-ящики должны быть обратимы.
- Принцип лавины: маленькие изменения на входе должны приводить к большим изменениям на выходе; в частности, изменение одного бита должно влиять на все биты результата. Для этого нужно потребовать выполнения следующих свойств:
  1. изменения одного бита на входе в S-ящик приводит к изменению по крайней мере *двух* битов на выходе из него;
  2. перетасовка битов должна быть устроена так, что биты на выходе из одного S-ящик должны попадать в *разные S-ящики* в следующем раунде.

2.5.10. Сеть Фейстеля. Кроме substitution-permutation сетей, существуют и другие способы конструирования псевдослучайных перестановок. Один из них — **сеть Фейстеля**. Основное преимущество сети Фейстеля состоит в том, что она позволяет сооружать обратимую функцию из необратимых компонент. Опишем i-й раунд действия сети Фейстеля. Аргумент (длины n) разбивается на две половины, L_{i-1} и R_{i-1}, длиной по $n/2$ бит. Раунд-функция f_i действует из $\{0,1\}^{n/2}$ в $\{0,1\}^{n/2}$ — но она не должна быть биективной. Результатом действия i-го раунда является строка длины n, полученная конкатенацией строк L_i и R_i длины $n/2$. где $L_i = R_{i-1}$ и $R_i = L_{i-1} \oplus f_i(R_{i-1})$. Таким образом, сеть Фейстеля с t раундами получает на вход строку из n бит, интерпретирует ее как (L_0, R_0), выполняет t раз указанную операцию, и выводит (L_t, R_t).

Раунд-функции f_i зависят от ключа; точнее, i-ая раунд-функция f_i зависит от i-го подключа k_i. Более формально, мы должны задать (открытую) функцию $\widehat{f}_i$ для каждого раунда $i = 1, \ldots, t$, и положить $f_i(x) = \widehat{f}_i(k_i, x)$.

Основное свойство сети Фейстеля состоит в том, что она обратима независимо от того, какие функции f_i мы выбрали. Покажем, как обратить i-й раунд: если нам дали пару (L_i, R_i), положим $R_{i-1} = L_i$ и $L_{i-1} = R_i \oplus f_i(R_{i-1})$.

В качестве примера упомянем, что стандарт шифрования данных DES является сетью Фейстеля с тщательно подобранными S-ящиками.

Еще одно применение сети Фейстеля — построение сильных псевдослучайных перестановок из псевдослучайных. Пусть F — псевдослучайная перестановка. Рассмотрим сеть Фейстеля $F^{(1)}$ с одним раундом, в которой функция f_1 равна F_k. Если длина ключа k равна n, то полученная сеть является перестановкой на $\{0,1\}^{2n}$. Нетрудно видеть, что это не псевдослучайная перестановка: первые n бит результата совпадают с последними n битами аргумента. Рассмотрим теперь такую сеть Фейстеля $F^{(2)}$ с двумя раундами; определим раунд-функции $f_1 = F_{k_1}$, $f_2 = F_{k_2}$ для ключа $k = k_1||k_2$ длины $2n$. Оказывается, полученная перестановка на $\{0,1\}^{2n}$ также не псевдослучайна. Повторим процедуру еще раз и рассмотрим сеть Фейстеля $F^{(3)}$с тремя раундами, в которых $f_1 = F_{k_1}$, $f_2 = F_{k_2}$, $f_3 = F_{k_3}$ для ключа $k = k_1||k_2||k_3$ длины $3n$. Оказывается, получится псевдослучайная перестановка на $\{0,1\}^{2n}$. Однако, она не сильно псевдослучайна. Наконец, сеть Фейстеля $F^{(k_4)}$ с четырьмя раундами, определенная аналогичным образом, является сильно псевдослучайной перестановкой на строках длины $2n$ с ключом длины $4n$.

2.6 Аутентификация сообщений

2.6.1. Необходимость аутентификации. Еще один важный класс задач, для решения которых используется криптография — проверка целостности сообщений. Пока ни один из рассмотренных нами шифров не защищает от следующей атаки: хакер может перехватить сообщение от Алисы к Бобу, идущее по открытому каналу связи, изменить его и послать Бобу измененное сообщение, а Боб не узнает о том, что сообщение подменили. Иными словами, часто Боб, получив сообщение, хочет быть уверен, что это сообщение исходит именно от Алисы, и по дороге оно не изменилось. Отметим, что эта задача возникает совершенно независимо от рассмотренных ранее: иногда содержание сообщения не является секретом, но важна его целостность.

Рассмотренный в разделе 2.2 потоковый шифр никак не защищает нас от изменений сообщений. Напомним, что в этом шифре зашифрованное сообщение имеет вид $c = G(k) \oplus m$, где G — псевдослучайный генератор. Изменение одного бита в сообщении c эквивалентно изменению одного (того же!) бита в исходном сообщении; то же касается нескольких бит. То же рассуждение годится для блочных шифров OFB и CTR. Шифры ECB и CBC не сильно лучше: например, в шифре ECB изменение одного бита в i-м зашифрованном блоке влияет только на i-й блок исходного сообщения. В шифре CBC изменение i-го бита в первом блоке зашифрованного сообщения (IV) приводит лишь к изменению i-го бита в первом блоке исходного сообщения; то есть, с первым блоком можно делать что угодно, как и в потоковом шифре.

Вообще, все рассмотренные выши схемы шифрования обладают тем свойством, что в них *каждый* зашифрованный текст соответствует какому-то исходному сообщению.

2.6.2. Определение MAC. Опишем общую схему механизмов, которые позволяют определить, изменялось ли сообщение по пути от отправителя к получателю. Такой механизм называется **MAC (message authentication code — код аутентификации сообщений)**. Снова мы предполагаем, что у двух сторон есть общий секретный ключ k. Когда Алиса хочет послать Бобу сообщение m, она вычисляет **ярлык** t по сообщению m и ключу k, а затем посылает ярлык вместе с сообщением m Бобу. Алгоритм вычисления ярылка будет обозначаться через Mac; таким образом, Алиса вычисляет $t = \mathrm{Mac}_k(m)$ и посылает пару (m, t) Бобу. Получив (m, t), Боб запускает алгоритм проверки Vrfy и узнает, соответствует ли ярлык t сообщению m, или нет. Дадим строгое определение

Определение 2.6.2.1. Код аутентификации сообщений — это тройка $(\mathrm{Gen}, \mathrm{Mac}, \mathrm{Vrfy})$ полиномиальных алгоритмов, удовлетворяющих следующим условиям:

1. Алгоритм генерации ключа Gen принимает натуральное число n и выводит ключ k такой, что $|k| \geqslant n$.

2. Алгоритм генерации ярлыка Mac может быть рандомизированным; он принимает ключ k и сообщение $m \in \{0,1\}^*$ и выводит ярлык $t \leftarrow \mathrm{Mac}(k, m)$.

3. Алгоритм проверки Vrfy принимает на вход ключ k, сообщение m и ярлык t. Он выдает бит b; значение $b = 1$ соответствует результату «корректно», а $b = 0$ — результату «некорректно». Без ограничения общности можно считать, что алгоритм Vrfy детерминистский: $b = \mathrm{Vrfy}(k, m, t)$.

4. Условие согласованности состоит в том, что для всех n, для всех ключей k, которые может выдавать $\mathrm{Gen}(n)$, и для всех сообщений m выполнено $\mathrm{Vrfy}(k, m, \mathrm{Mac}(k, m)) = 1$.

5. Пусть $k \leftarrow \mathrm{Gen}(n)$ — некоторый ключ; зачастую алгоритм $\mathrm{Mac}(k, m)$ определен только для сообщений m длины $l(n)$, а $\mathrm{Vrfy}(k, m, t)$ выводит 0 для всех $m \notin \{0,1\}^{l(n)}$. В этом случае мы говорим, что $(\mathrm{Gen}, \mathrm{Mac}, \mathrm{Vrfy})$ — MAC *для сообщений фиксированной длины* $l(n)$.

Как и раньше, алгоритм $\mathrm{Gen}(n)$ в большинстве случаев выбирает случайную строку из $\{0,1\}^n$.

Для определения надежности кода аутентификации сообщений мы дадим хакеру доступ к оракулу, вычисляющему функцию $\mathrm{Mac}(k, -)$; хакер может отправить этому оракулу произвольное сообщение m и получить в ответ ярлык $t \leftarrow \mathrm{Mac}(k, m)$. Мы будем говорить, что хакер *сломал* наш MAC, если ему удалось предъявить пару (m, t) такую, что, во-первых, $\mathrm{Vrfy}(k, m, t) = 1$, а во-вторых, хакер ранее не запрашивал у оракула ярлык для сообщения m. Если MAC устойчив относительно такого рода атаки, мы говорим, что он обладает свойством *экзистенциальной неподделываемости относительно адаптивных chosen-message атак* — а для краткости, просто, что этот MAC *надежен*.

Таким образом, эксперимент по аутентификации сообщений $\mathrm{MacForge}_{\mathcal{A},S}(n)$ для кода аутентификации сообщений $S = (\mathrm{Gen}, \mathrm{Mac}, \mathrm{Vrfy})$, хакера $\mathcal{A}$ и значения параметра n устроен следующим образом.

1. Мы генерируем случайный ключ $k \leftarrow \mathrm{Gen}(n)$.

2. Хакер $\mathcal{A}$ получает параметр n и доступ к оракулу $\mathrm{Mac}(k, -)$; он выводит пару (m, t). Пусть Q — множество всех запросов, который $\mathcal{A}$ посылал оракулу.

3. Результат эксперимента равен 1 тогда и только тогда, когда $\mathrm{Vrfy}(k, m, t) = 1$ и $m \notin Q$.

Определение 2.6.2.2. Код аутентификации сообщений $S = (\mathrm{Gen}, \mathrm{Mac}, \mathrm{Vrfy})$ называется **надежным**, если для любого полиномиального хакера $\mathcal{A}$ существует пренебрежимо малая функция ε такая, что $\mathrm{P}(\mathrm{MacForge}_{\mathcal{A},S}(n) = 1) \leqslant \varepsilon(n)$.

2.6.3. MAC из псевдослучайной функции. Покажем, что по любой псевдослучайной функции F можно построить код аутентификации сообщений (фиксированной длины $l(n) = n$).

- Алгоритм $\mathrm{Gen}(n)$ выдает случайную равномерно распределенную строку k из $\{0,1\}^n$.

- Алгоритм Mac по ключу $k \in \{0,1\}^n$ и сообщению $m \in \{0,1\}^n$ выдает ярлык $t = F_k(m)$.
- Алгоритм Vrfy по ключу $k \in \{0,1\}^n$, сообщению $m \in \{0,1\}^n$ и ярлыку $t \in \{0,1\}^n$ выдает 1 если и только если $t = F_k(m)$.

Теорема 2.6.3.1. *Если F — псевдослучайная функция, то схема шифрования, описанная выше, является надежной.*

Доказательство. Пусть $\mathcal{A}$ — полиномиальный хакер. Положим

$$\varepsilon(n) = P(\text{MacForge}_{\mathcal{A},S}(n) = 1).$$

Рассмотрим вспомогательный код аутентификации сообщений $\widetilde{S} = (\widetilde{\text{Gen}}, \widetilde{\text{Mac}}, \widetilde{\text{Vrfy}})$, который отличается от S использованием случайной равномерно распределенной функции f вместо функции F_k. Значения функции f равномерно распределены в $\{0,1\}^n$ независимо друг от друга. Поэтому с точки зрения хакера $\mathcal{A}$ для $m \notin Q$ значение $t = f(m)$ равномерно распределено на $\{0,1\}^n$. Поэтому

$$P(\text{MacForge}_{\mathcal{A},S}(n) = 1) \leqslant 2^{-n}.$$

Построим полиномиальный алгоритм D, отличающий функцию от псевдослучайной. Для достижения этой цели алгоритм D будет имитировать эксперимент по аутентификации сообщений для хакера $\mathcal{A}$; если хакеру удастся построить корректную пару (m,t), то D будет считать, что он имеет дело с псевдослучайной функцией, а если нет — то с равномерно распределенной случайной. Более строго, D получает на вход число n и доступ к оракулу $\mathcal{O}\colon \{0,1\}^n \to \{0,1\}^n$. Он работает следующим образом:

1. D запускает $\mathcal{A}(n)$. Каждый раз, когда алгоритм $\mathcal{A}$ посылает запрос к своему оракулу с сообщением m, D в свою очередь посылает это сообщение m своему оракулу $\mathcal{O}$ и получает в ответ t; этот ответ он пересылает хакеру $\mathcal{A}$.
2. В итоге $\mathcal{A}$ выводит пару (m,t). После этго D посылает оракулу $\mathcal{O}$ запрос m и получает ответ t'. Если $t = t'$ и при этом $\mathcal{A}$ никогда не посылал своему оракулу запрос m, алгоритм D выводит 1; иначе, он выводит 0.

Заметим, что если алгоритму D в качестве оракула попалась псевдослучайная функция, то с точки зрения $\mathcal{A}$ происходит эксперимент $\text{MacForge}_{\mathcal{A},S}(n)$, и вывод D равен 1 тогда и только тогда, когда этот эксперимент завершается успехом. Поэтому

$$P(D^{F_k}(n) = 1) = P(\text{MacForge}_{\mathcal{A},S}(n) = 1) = \varepsilon(n),$$

где вероятность соответствует равномерном распределению $k \leftarrow \{0,1\}^n$. Если же оракул D вычисляет случайную функцию, то с точки зрения $\mathcal{A}$ происходит эксперимент $\text{MacForge}_{\mathcal{A},\widetilde{S}}(n)$, и вывод D снова равен 1 тогда и только тогда, когда этот эксперимент завершается успехом. Поэтому

$$P(D^f(n) = 1) = P(\text{MacForge}_{\mathcal{A},\widetilde{S}}(n) = 1) \leqslant 2^{-n},$$

где вероятность соответствует равномерному распределению $f \leftarrow \text{Func}_n$. Поэтому

$$P(D^{F_k}(n) = 1) - P(D^f(n) = 1) \geqslant \varepsilon(n) - 2^{-n}.$$

По определению псевдослучайной функции правая часть пренебрежимо мала; потому и $\varepsilon(n)$ пренебрежимо мала. □

3 Шифрование с открытым ключом

3.1 Основные протоколы

3.1.1. Асимметричное шифрование. В 1976 году Уитфилд Диффи и Мартин Хеллман опубликовали статью, которая совершила революцию в криптографии. До их работы шифрование всегда опиралось на общий закрытый ключ. Диффи и Хеллман обратили внимание на то, что мир вокруг нас часто асимметричен: есть действия, которые легко совершить, но трудно обратить. Поэтому возможно создать схему шифрования, которая не опирается на общий секретный ключ, но такую, в которой шифрование совершается «легко», но обратную процедуру — дешифровку — совершить сложно без некоторого знания, которым обладает только получатель.

Представим себе схему шифрования, которая использует два ключа вместо одного: один используется для шифрования теми, кто отсылает сообщения, а второй — для дешифровки получателем. Более того, зашифрованные сообщения не может прочитать даже тот, кто знает ключ для шифрования (но не знает ключа для дешифрования). Такие схемы называются **асимметричными**, или **схемами шифрования с открытым ключом** (в отличие от *симметричных* схем шифрования с закрытым ключом). В такой схеме ключ для шифрования называется **открытым ключом**, а ключ для дешифровки — **закрытым ключом**. Открытый ключ не является секретом, и получатель сообщает его каждому, от кого он хочет получать сообщения. Для дешифровки используется закрытый ключ, который получатель хранит в тайне.

Диффи и Хеллман предложили три протокола асимметричной криптографии: шифрование с открытым ключом, цифровые подписи (аналогичные кодам аутентификации сообщений), и интерактивный обмен ключами. Цифровые подписи отличаются от MAC тем, что аутентичность сообщений может проверить любой, кому известен открытый ключ отправителя. Интерактивный обмен ключами — это метод, благодаря которому две стороны, у которых нет никакой общей секретной информации, могут договориться об общем секретном ключе по открытым каналам связи так, что подслушивающий не узнает ничего об этом ключе (хотя получает все сообщения, которые посылаются между сторонами). Конечно, в наивном описании такой системы остается проблема аутентификации: мы должны быть уверены, что договариваемся именно с той стороной, с которой нужно, а не с хакером, выдающим себя за другого.

Диффи и Хеллман описали в своей статье только алгоритм интерактивного обмена ключами, а через год Рон Ривест, Ади Шамир и Леонард Эйдельман предложили алгоритмы RSA для шифрования с открытым ключом и цифровой подписи. Чуть позднее Эль-Гамаль предложил алгоритмы для шифрования с открытым ключом и цифровой подписи, основанные на оригинальной идее Диффи–Хеллмана (см. раздел 3.3.4).

3.1.2. Интерактивный обмен ключами. Опишем общую задачу интерактивного обмена ключами. Мы хотим, чтобы Алиса и Боб, начав с параметра n, совершили некоторый операции в соответствии с протоколом Π, в результате которого они получают ключи $k_A, k_B \in \{0,1\}^n$ соответственно. Требование корректности состоит в том, что $k_A = k_B$. Надежность протокола означает, что общий ключ, полученный Алисой и Бобом, неизвестен подслушивающему хакеру. Для формализации этого потребуем, чтобы хакер не смог отличить этот ключ от случайной строки длины n.

Итак, пусть Π — протокол обмена ключами, $\mathcal{A}$ — хакер, n — натуральное число. Эксперимент $\mathrm{KE}^{\mathrm{eav}}_{\mathcal{A},\Pi}(n)$ заключается в следующем:

1. Две стороны, обладающие параметром n, выполняют протокол Π. Обозначим через trans стенограмму этого протокола, то есть, последовательность всех сообщений, которые стороны посылали друг другу. Результатом работы протокола является ключ $k = k_A = k_B$, который получает каждая сторона.

2. Выбирается случайный бит $b \leftarrow \{0,1\}$. Если $b = 0$, выберем случайную равномерно распределенную строку $k' \leftarrow \{0,1\}^n$; если $b = 1$, положим $k' = k$.

3. Хакер $\mathcal{A}$ получает стенограмму trans и строку k', и выводит бит b'.

4. Результат работы эксперимента равен 1, если $b' = b$ (и мы говорим, что эксперимент завершился успехом), и 0 в противоположном случае.

Определение 3.1.2.1. Протокол обмена ключами называется **надежным**, если для любого полиномиального хакера существует пренебрежимо малая функция ε такая, что

$$P(KE^{eav}_{\mathcal{A},\Pi}(n) = 1) \leqslant \frac{1}{2} + \varepsilon(n).$$

3.1.3. Протокол Диффи–Хеллмана. Приведем протокол обмена ключами, содержащийся в статье Диффи и Хеллмана. Пусть $\mathcal{G}$ — полиномиальный [вероятностный] алгоритм, который по натуральному числу n выводит описание циклической группы G, порядок этой группы $q = |G|$ (где длина двоичной записи q равна n), и образующую g этой группы. Мы требуем, чтобы групповая операция в G вычислялась за полиномиальное время. Протокол Диффи–Хеллмана работает так:

1. Алиса запускает $\mathcal{G}(n)$ и получает (G, q, g).

2. Алиса выбирает x случайным образом из $\mathbb{Z}/q\mathbb{Z}$ и вычисляет $h_1 = g^x$.

3. Алиса посылает (G, q, g, h_1) Бобу.

4. Боб получает (G, q, g, h_1); выбирает y случайным образом из $\mathbb{Z}/q\mathbb{Z}$ и вычисляет $h_2 = g^y$. Боб посылает h_2 Алисе и выводит ключ $k_B = h_1^y$.

5. Алиса получает h_2 и выводит ключ $k_A = h_2^x$.

Нетрудно видеть, что этот протокол корректен: Боб выводит $k_B = h_1^y = (g^x)^y = g^{xy}$, а Алиса выводит $k_A = h_2^x = (g^y)^x = g^{xy}$, и потому $k_A = k_B$. Впрочем, полученный ключ является не строкой длины n, а элементом группы G; мы вернемся к этой проблеме позже.

Почему этот протокол надежен? Во всяком случае, задача *дискретного логарифмирования* должна быть сложной: если бы можно было по g и h_1 вычислить $x = \log_g h_1$, то есть, $x \in \mathbb{Z}_q$ такое, что $g^x = h_1$, то хакер мог бы вычислить $k_A = h_2^x$, как Алиса. Однако, сложности дискретного логарифмирования недостаточно; надежность указанного протокола опирается на более сильное требование, которое называется *сложностью задачи различения Диффи–Хеллмана (DDH)*.

Пусть G — циклическая группа, $g \in G$ — ее образующая. Для элементов $h_1, h_2 \in G$ положим $DH_g(h_1, h_2) = g^{\log_g h_1 \cdot \log_g h_2}$. То есть, если $h_1 = g^x$ и $h_2 = g^y$, то $DH_g(h_1, h_2) = g^{xy} = h_1^y = h_2^x$. Задача DDH состоит в том, чтобы отличить $DH_g(h_1, h_2)$ от случайного элемента группы для случайно выбранных h_1, h_2.

Определение 3.1.3.1. Будем говорить, что **задача DDH сложна относительно** $\mathcal{G}$, если для любого полиномиального алгоритма $\mathcal{A}$ существует пренебрежимо малая функция ε такая, что

$$|\mathrm{P}(\mathcal{A}(G, q, g, g^x, g^y, g^z) = 1) - \mathrm{P}(\mathcal{A}(G, q, g, g^x, g^y, g^{xy}) = 1)| \leqslant \varepsilon(n),$$

где вероятности берутся по всем экспериментам, в которых $\mathcal{G}(n)$ выводит (G, q, g), и по всем случайным выборам $x, y, z \in \mathbb{Z}/q\mathbb{Z}$ для каждого вывода $\mathcal{G}(n)$.

Есть мнение (не доказанное), что задача DDH сложно относительно следующего алгоритма генерации групп $\mathcal{G}_1$. Рассмотрим группу $(\mathbb{Z}/p\mathbb{Z})^*$. Ее порядок равен $p - 1$, что не может быть простым числом; однако, возможно, что $p - 1 = 2q$, где q простое (такие простые числа p называются **сильными простыми**). Рассмотрим в $(\mathbb{Z}/p\mathbb{Z})^*$ подгруппу, состоящую из *квадратичных вычетов*, то есть, из элементов вида x^2, $x \in (\mathbb{Z}/p\mathbb{Z})^*$. В такой группе ровно $(p-1)/2 = q$ элементов. Поэтому это циклическая группа, и любой ее нетривиальный элемент является образующей.

Алгоритм $\mathcal{G}_1$ выглядит так: получив n, он генерирует случайное сильное простое число p длины $(n + 1)$ (это возможно, как мы узнаем позже). Пусть $q = (p - 1)/2$; выберем образом $x \in (\mathbb{Z}/p\mathbb{Z})^*$ так, чтобы $x \neq \pm 1$, и положим $g = x^2$. Вернем тройку (p, q, g); здесь число p служит описанием группы квадратичных вычетов по модулю p.

Недостаток протокола Диффи–Хеллмана состоит в том, что итоговым ключом в нем является элемент некоторой группы, а не строка длины n. Как правило, если элемент группы представить (некоторым естественным образом) в виде битовой строки, ее очень легко отличить от случайной строки. Определение 3.1.2.1 же требует, чтобы ключ был неотличим от случайной строки. Определим модифицированный эксперимент $\widehat{\mathrm{KE}^{\mathrm{eav}}}_{\mathcal{A},S}$, в котором требуется отличить сгенерированный ключ не от случайной строки, а от случайного элемента группы; то есть, в случае $b = 0$ мы выбираем k' равномерно из группы G.

Теорема 3.1.3.2. *Если задача DDH сложна относительно $\mathcal{G}$, то протокол обмена ключами S надежен (по отношению к модифицированному эксперименту $\widehat{\mathrm{KE}^{\mathrm{eav}}}_{\mathcal{A},S}$).*

Доказательство. Поскольку $\mathrm{P}(b = 0) = \mathrm{P}(b = 1) = 1/2$, то

$$\mathrm{P}(\widehat{\mathrm{KE}^{\mathrm{eav}}}_{\mathcal{A},S}(n) = 1) = \frac{1}{2} \cdot \mathrm{P}(\widehat{\mathrm{KE}^{\mathrm{eav}}}_{\mathcal{A},S}(n) = 1 \mid b = 1) + \frac{1}{2} \cdot \mathrm{P}(\widehat{\mathrm{KE}^{\mathrm{eav}}}_{\mathcal{A},S}(n) = 1 \mid b = 0).$$

Напомним, что в эксперименте $\widehat{\mathrm{KE}^{\mathrm{eav}}}_{\mathcal{A},S}(n)$ хакер $\mathcal{A}$ получает набор (G, q, g, h_1, h_2, k'), где все данные, кроме k', составляют стенограмму работы протокола, а k' равен либо ключу g^{xy} (если $b = 1$), либо случайному элементу группы (если $b = 0$). Различение между этими случаями в точности эквивалентно решению задачу DDH. Поэтому

$$\begin{aligned}
\mathrm{P}(\widehat{\mathrm{KE}^{\mathrm{eav}}}_{\mathcal{A},S}(n) = 1) &= \frac{1}{2} \cdot \mathrm{P}(\mathcal{A}(G, g, q, g^x, g^y, g^{xy}) = 1) + \frac{1}{2} \cdot \mathrm{P}(\mathcal{A}(G, g, q, g^x, g^y, g^z) = 0) \\
&= \frac{1}{2} \cdot \mathrm{P}(\mathcal{A}(G, g, q, g^x, g^y, g^{xy}) = 1) + \frac{1}{2} \cdot (1 - \mathrm{P}(\mathcal{A}(G, g, q, g^x, g^y, g^z) = 1)) \\
&\leqslant \frac{1}{2} + \frac{1}{2} \cdot |\mathrm{P}(\mathcal{A}(G, g, q, g^x, g^y, g^{xy}) = 1) - \mathrm{P}(\mathcal{A}(G, g, q, g^x, g^y, g^z) = 1)|,
\end{aligned}$$

где вероятности берутся по всем выводам (G, q, g) алгоритма $\mathcal{G}(n)$ и по всем x, y, z, равномерно распределенным по $\mathbb{Z}/q\mathbb{Z}$. Если задача DDH сложна относительно $\mathcal{G}$, то существует пренебрежимо малая функция ε такая, что модуль разности вероятностей в полученном выражении не превосходит $\varepsilon(n)$. Поэтому

$$\mathrm{P}(\widehat{\mathrm{KE}^{\mathrm{eav}}}_{\mathcal{A},S}(n) = 1) \leqslant \frac{1}{2} + \frac{1}{2} \cdot \varepsilon(n).$$

□

3.1.4. Эллиптические кривые. Еще один важный класс групп, используемых в криптографии — это *группы точек эллиптических кривых*. Для них пока не найдено субэкспоненциального алгоритма решения задачи дискретного логарифмирования (в отличие от подгрупп $(\mathbb{Z}/p\mathbb{Z})^*$. Пусть p — простое число, большее 3. Рассмотрим пары (x, y) элементов $\mathbb{Z}/p\mathbb{Z}$, удовлетворяющие уравнению $y^2 = x^3 + Ax + B$ в $\mathbb{Z}/p/Z$, где $A, B \in \mathbb{Z}/p\mathbb{Z}$ — некоторые константы. Потребуем также, чтобы $4A^3 + 27B^2$ не равнялось 0 по модулю p. Просвещенный читатель заметит, что речь идет о точках алгебраического многообразия E, заданного уравнением $y^2 = x^3 + Ax + B$, над конечным полем из p элементов, а условие $4A^3 + 27B^2 \neq 0$ гарантирует гладкость этого многообразия во всех точках. Замыкание такого многообразия в проективном пространстве $\mathbb{P}^2$ называется *эллиптической кривой*.

Итак, пусть $\widehat{E}(\mathbb{Z}/p\mathbb{Z}) = \{(x, y) \in (\mathbb{Z}/p\mathbb{Z})^2 \mid y^2 = x^3 + Ax + B\}$, и положим $E(\mathbb{Z}/p\mathbb{Z}) = \widehat{E}(\mathbb{Z}/p\mathbb{Z}) \cup \{\mathcal{O}\}$, где $\mathcal{O}$ — специальный элемент, который будет играть роль нейтрального элемента в нашей группе. Элементы $E(\mathbb{Z}/p\mathbb{Z})$ называются **точками эллиптической кривой** E, а $\mathcal{O}$ — **точкой на бесконечности**. Оказывается, на множестве $E(\mathbb{Z}/p\mathbb{Z})$ можно ввести операцию, превращающую его в абелеву группу. На практике удобно подбирать параметры A и B так, чтобы получилась группа простого порядка, которая автоматически является циклической.

3.1.5. Шифрование с открытым ключом.

Определение 3.1.5.1. Схема шифрования с открытым ключом — это тройка вероятностных полиномиальных алгоритмов $(\mathrm{Gen}, \mathrm{Enc}, \mathrm{Dec})$ такая, что выполняются следующие условия.

1. Алгоритм генерации ключа Gen принимает на вход натуральное число n и выводит пару ключей (pk, sk), где pk называется **открытым ключом**, а sk — **закрытым (или секретным) ключом**. Мы будем предполагать, что ключи pk и sk имеют длину хотя бы n.

2. Алгоритм шифрования End принимает открытый ключ pk и сообщение $m \in \mathcal{M}$ (где $\mathcal{M}$ может зависеть от pk), и выводит зашифрованное сообщение $c \leftarrow \mathrm{End}(pk, m)$.

3. Алгоритм дешифровки Dec принимает закрытый ключ sk и зашифрованный текст c, и выводит либо сообщение m, либо символ ошибки $\perp$. Мы будем считать, что алгоритм Dec детерминистский: $m = \mathrm{Dec}(sk, c)$.

4. Условие согласованности:
$$\mathrm{Dec}(sk, \mathrm{Enc}(pk, m)) = m$$
с вероятностью, отличающейся от 1 на пренебрежимо малую величину, где вероятность берется по всем парам (pk, sk), которые выводит алгоритм $\mathrm{Gen}(n)$ и по всем «бросаниям монетки» в алгоритме Enc.

3.2 Надежность шифрования с открытым ключом

3.2.1. Определение надежности. Пусть $S = (\mathrm{Gen}, \mathrm{Enc}, \mathrm{Dec})$ — схема шифрования с открытым ключом. Определим эксперимент по взлому, аналогичный эксперименту, описанному в разделе 2.1.6.

1. Мы запускаем алгоритм $\mathrm{Gen}(n)$ и получаем пару ключей (pk, sk).

2. Сообщаем хакеру $\mathcal{A}$ открытый ключ pk. Он выводит пару сообщений m_0, m_1 одной длины.

3. Мы выбираем случайный бит b из $\{0,1\}$, вычисляем зашифрованное сообщение $c \leftarrow \mathrm{Enc}(pk, m_b)$ и отправляем его хакеру.

4. $\mathcal{A}$ выводит бит b'.

5. Результат эксперимента равен 1 (*эксперимент завершился успехом*), если $b' = b$, и 0 в противном случае.

Определение 3.2.1.1. Схема шифрования с открытым ключом $S = (\mathrm{Gen}, \mathrm{Enc}, \mathrm{Dec})$ называется **надежной относительно подслушивания**, если для любого полиномиального хакера $\mathcal{A}$ существует пренебрежимо малая функция ε такая, что

$$P(\mathrm{PubK}^{eav}_{\mathcal{A},S}(n) = 1) \leqslant \frac{1}{2} + \varepsilon(n).$$

Подчеркнем еще раз, что в описанном эксперименте хакеру $\mathcal{A}$ известен открытый ключ pk. На самом деле это означает, что хакеру предоставлен доступ к оракулу шифрования: действительно, зная pk, хакер может запустить [общеизвестный] алгоритм Enc и вычислить $\mathrm{Enc}(pk, m)$ для любого сообщения m. Из этого следует, что определение 3.2.1.1 надежности относительно подслушивания эквивалентно определению устойчивости относительно chosen plaintext-атак.

А именно, определим эксперимент $\mathrm{PubK}^{cp}_{\mathcal{A},S}(n)$ следующим образом.

1. Мы запускаем алгоритм $\mathrm{Gen}(n)$ и получаем пару ключей (pk, sk).

2. Сообщаем хакеру $\mathcal{A}$ открытый ключ pk и доступ к оракулу, вычисляющему функцию $\mathrm{Enc}(pk, -)$. Хакер выводит пару сообщений m_0, m_1 одной длины.

3. Мы выбираем случайный бит b из $\{0,1\}$, вычисляем зашифрованное сообщение $c \leftarrow \mathrm{Enc}(pk, m_b)$ и отправляем его хакеру.

4. $\mathcal{A}$ продолжает иметь доступ к оракулу $\mathrm{Enc}(pk, -)$ и выводит бит b'.

5. Результат эксперимента равен 1 (*эксперимент завершился успехом*), если $b' = b$, и 0 в противном случае.

Определение 3.2.1.2. Схема шифрования с открытым ключом $S = (\mathrm{Gen}, \mathrm{Enc}, \mathrm{Dec})$ называется **надежной в присутствии chosen plaintext-атак**, (или **устойчивой относительно chosen plaintext-атак**), если для любого полиномиального хакера $\mathcal{A}$ существует пренебрежимо малая функция ε такая, что

$$P(\mathrm{PubK}^{cp}_{\mathcal{A},S}(n) = 1) \leqslant \frac{1}{2} + \varepsilon(n).$$

Таким образом, если схема шифрования S надежна относительно подслушивания, то она устойчива относительно chosen plaintext-атак.

В определении 3.2.1.1 мы могли бы потребовать, чтобы для *любого* хакера $\mathcal{A}$ выполнялось равенство $P(\mathrm{PubK}^{eav}_{\mathcal{A},S}(n) = 1) = 1/2$ (аналогично понятию совершенной секретности, см. раздел 2.1.6). В случае шифрования с открытым ключом совершенная секретность недостижима: зная pk и $c \leftarrow \mathrm{Enc}(pk, m)$, можно определить m с вероятностью 1, если ресурсы хакера неограниченны.

В разделе 2.5.5 мы заметили, что детерминистское шифрование не может быть устойчиво относительно chosen plaintext-атак. Ввиду эквивалентности надежности относительно chosen plaintext-атак и надежности относительно подслушивания, получаем, что детерминистская схема шифрования с открытым ключом не может быть надежной относительно подслушивания.

3.2.2. Шифрование нескольких сообщений. Рассмотрим теперь, что происходит при шифровании нескольких сообщений. Для шифрования с закрытым ключом, как мы знаем из раздела 2.5.7, надежность шифрования нескольких сообщений следует из надежности шифрования одного, если речь идет о chosen plaintext-атаках. В случае открытого ключа, разумеется, достаточно говорить о надежности относительно подслушивания. Опишем эксперимент $\mathrm{PubK}^{\mathrm{mult}}_{\mathcal{A},S}(n)$ по взлому в присутствии подслушивающего для нескольких сообщений.

1. Мы запускаем алгоритм $\mathrm{Gen}(n)$ и получаем пару ключей (pk, sk).
2. Сообщаем хакеру $\mathcal{A}$ открытый ключ pk. Хакер выводит пару наборов сообщений $M_0 = (m_0^{(1)}, \ldots, m_0^{(t)})$ и $M_1 = (m_1^{(1)}, \ldots, m_1^{(t)})$.
3. Мы выбираем случайный бит b из $\{0, 1\}$, вычисляем зашифрованные сообщения $c^{(i)} \leftarrow \mathrm{Enc}(pk, m_b^{(i)})$ и отправляем набор $C = (c^{(1)}, \ldots, c^{(t)})$ хакеру.
4. $\mathcal{A}$ выводит бит b'.
5. Результат эксперимента равен 1 (*эксперимент завершился успехом*), если $b' = b$, и 0 в противном случае.

Определение 3.2.2.1. Схема шифрования с открытым ключом $S = (\mathrm{Gen}, \mathrm{Enc}, \mathrm{Dec})$ называется **надежной для шифрования нескольких сообщений относительно подслушивания**, если для любого полиномиального хакера $\mathcal{A}$ существует пренебрежимо малая функция ε такая, что

$$\mathrm{P}(\mathrm{PubK}^{\mathrm{mult}}_{\mathcal{A},S}(n) = 1) \leqslant \frac{1}{2} + \varepsilon(n).$$

Теорема 3.2.2.2. *Если схема шифрования S с открытым ключом надежна относительно подслушивания, то она надежна для шифрования нескольких сообщений относительно подслушивания.*

Доказательство. Мы докажем эту теорему только для случая, когда хакер всегда выводит два набора по два сообщения (то есть, $t = 2$). Общий случай при желании рассматривается аналогично (хотя нужно учесть, что t может зависеть от n и от pk). Пусть $\mathcal{A}$ — полиномиальный хакер; рассмотрим эксперимент $\mathrm{PubK}^{\mathrm{mult}}_{\mathcal{A},S}(n)$. В этом эксперименте хакер получает либо $C = (\mathrm{Enc}(pk, m_0^{(1)}), \mathrm{Enc}(pk, m_0^{(2)}))$, либо $C = (\mathrm{Enc}(pk, m_1^{(1)}), \mathrm{Enc}(pk, m_1^{(2)}))$. Заметим, что

$$\mathrm{P}(\mathrm{PubK}^{\mathrm{mult}}_{\mathcal{A},S}(n) = 1) = \frac{1}{2}\cdot\mathrm{P}(\mathcal{A}(\mathrm{Enc}(pk, m_0^{(1)}), \mathrm{Enc}(pk, m_0^{(2)})) = 0) + \frac{1}{2}\cdot\mathrm{P}(\mathcal{A}(\mathrm{Enc}(pk, m_1^{(1)}), \mathrm{Enc}(pk, m_1^{(2)})) = 1).$$

Посмотрим, что произойдет, если отправить хакеру пару $(\mathrm{Enc}(pk, m_0^{(1)}), \mathrm{Enc}(pk, m_1^{(2)}$. Мы утверждаем, что существует пренебрежимо малая функция φ такая, что

$$\frac{1}{2}\cdot\mathrm{P}(\mathcal{A}(\mathrm{Enc}(pk, m_0^{(1)}), \mathrm{Enc}(pk, m_0^{(2)})) = 0) + \frac{1}{2}\cdot\mathrm{P}(\mathcal{A}(\mathrm{Enc}(pk, m_0^{(1)}), \mathrm{Enc}(pk, m_1^{(2)})) = 1) \leqslant \frac{1}{2} + \varphi(n).$$

Для доказательства этого построим полиномиального хакера $\mathcal{A}'$, который занимается прохождением эксперимента по взлому относительно подслушивания для *одного сообщения*. А именно, хакер $\mathcal{A}'$, получив открытый ключ pk, запускает хакера $\mathcal{A}$, дает ему этот ключ и получает два набора $M_0 = (m_0^{(1)}, m_0^{(2)})$, $M_1 = (m_1^{(1)}, m_1^{(2)})$. После этого $\mathcal{A}'$ возвращает в эксперимент пару сообщений (m_0^2, m_1^2), и получает зашифрованное сообщение $c^{(2)}$. Далее он вычисляет $c^{(1)} \leftarrow \mathrm{Enc}(pk, m_0^1)$, отдает хакеру $\mathcal{A}$ набор $(c^{(1)}, c^{(2)})$, и выводит тот же бит b', что получает от хакера $\mathcal{A}$.

Каков результат эксперимента $\mathrm{PubK}^{\mathrm{eav}}_{\mathcal{A}',S}(n)$? Если $b = 0$, то хакер $\mathcal{A}'$ получает $\mathrm{Enc}(pk, m_0^{(2)})$, и

$$P(\mathcal{A}'(\mathrm{Enc}(pk, m_0^{(2)}) = 0) = P(\mathcal{A}(\mathrm{Enc}(pk, m_0^{(1)}), \mathrm{Enc}(pk, m_0^{(2)})) = 0).$$

Если же $b = 1$, то хакер $\mathcal{A}'$ получает $\mathrm{Enc}(pk, m_1^{(2)})$, и

$$P(\mathcal{A}'(\mathrm{Enc}(pk, m_1^{(2)}) = 1) = P(\mathcal{A}(\mathrm{Enc}(pk, m_0^{(1)}), \mathrm{Enc}(pk, m_1^{(2)})) = 0).$$

По предположению надежности схемы S для шифрования одного сообщения существует пренебрежимо малая функция φ такая, что

$$??? \leqslant \frac{1}{2} + \varphi(n).$$

Аналогично показывется, что существует пренебрежимо малая функция ψ такая, что

$$\frac{1}{2} \cdot P(\mathcal{A}(\mathrm{Enc}(pk, m_0^{(1)}), \mathrm{Enc}(pk, m_1^{(2)})) = 0) + \frac{1}{2} \cdot P(\mathcal{A}(\mathrm{Enc}(pk, m_1^{(1)}), \mathrm{Enc}(pk, m_1^{(2)})) = 1) \leqslant \frac{1}{2} + \psi(n).$$

Складывая два полученных неравенства, получаем, что

$$\frac{1}{2} + P(\mathrm{PubK}^{\mathrm{mult}}_{\mathcal{A},S}(n) = 1) \leqslant 1 + \varepsilon(n)$$

для пренебрежимо малой функции $\varepsilon = \varphi + \psi$. Отсюда следует, что

$$P(\mathrm{PubK}^{\mathrm{mult}}_{\mathcal{A},S}(n) = 1) \leqslant \frac{1}{2} + \varepsilon(n).$$

□

3.2.3. Гибридное шифрование. Один из основных недостатков шифрования с открытым ключом состоит в том, что алгоритмы зашифровки и дешифровки работают на 2–3 порядка медленнее, чем алгоритмы для шифрования с закрытым ключом. Поэтому для зашифровки длинных сообщений имеет смысл применять следующую гибридную схему: зашифровать все сообщение посредством закрытого ключа, а сам этот ключ зашифровать с помощью криптографии с открытым ключом.

А имеено, пусть $S = (\mathrm{Gen}, \mathrm{Enc}, \mathrm{Dec})$ — схема шифрования с открытым ключом, а $S' = (\mathrm{Gen}', \mathrm{Enc}', \mathrm{Dec}')$ — схема шифрования с закрытым ключом. Опишем гибридную схему шифрования, основанную на S, S'. Ее алгоритм генерации ключа просто запускает алгоритм Gen и возвращает полученную пару (pk, sk) из закрытого и открытого ключа. Для зашифровки сообщения m сгенерируем закрытый ключ k посредством алгоритма Gen' и зашифруем его с помощью открытого ключа pk: $c_1 = \mathrm{Enc}(pk, k)$. После этого зашифруем m посредством схемы S': $c_2 = \mathrm{Enc}'(k, m)$. Итоговое зашифрованное гибридным образом сообщение — это пара (c_1, c_2). Понятно, как расшифровать ее: сначала вычислим $k = \mathrm{Dec}(sk, c_1)$, а затем и $m = \mathrm{Dec}'(k, c_2)$.

Теорема 3.2.3.1. *Если S — схема шифрования с открытым ключом, устойчивая относительно chosen plaintext-атак, и S' — схема шифрования с закрытым ключом, устойчивая относительно подслушивания, то описанная выше гибридная схема является схемой шифрования с открытым ключом, устойчивой относительно chosen plaintext-атак.*

Мы не будем доказывать эту теорему.

3.3 RSA

3.3.1. Наивная схема RSA. Сначала мы опишем наивный вариант шифрования RSA. Сразу подчеркнем, что этот вариант весьма ненадежен, и не должен применяться на практике (хотя именно он описан во многих учебниках).

Пусть GenRSA — полиномиальный (как всегда, по n) алгоритм, который по числу n выводит число N, являющееся произведением двух простых чисел, каждое из которых имеет длину n бит, вместе с целыми числами e, d такими, что $ed \equiv 1 \pmod{\varphi(N)}$. Такой алгоритм несложно построить, если мы умеем генерировать простые числа длины n. Действительно, сгенерировав два таких простых числа p и q, положим $N = pq$. Тогда $\varphi(N) = (p-1)(q-1)$, и мы можем выбрать e такое, что $\gcd(e, \varphi(N)) = 1$. После этого несложно вычислить $d \equiv e^{-1} \pmod{\varphi(N)}$.

Отметим, что на практике генерация простого числа длины n может с небольшой вероятностью вернуть *составное* число (поскольку используются эффективные *вероятностные* тесты на простоту). Поэтому мы допускаем, что работа алгоритм GenRSA может завершиться неудачей, но эта вероятность пренебрежимо мала и мы не будем обращать на нее внимание.

Опишем «наивную» схему шифрования RSA с открытым ключом.

- Алгоритм Gen, получив на вход параметр n, запускает GenRSA(n) и получает числа N, e, d. Открытым ключом служит пара (N, e), а закрытым — пара (N, d).
- Алгоритм Gen по открытому ключу $pk = (N, e)$ и сообщению $m \in (\mathbb{Z}/N\mathbb{Z})^*$ вычисляет $c = (m^e \bmod N)$.
- Алгоритм Dec, получив закрытый ключ $sk = (N, d)$ и зашифрованное сообщение $c \in (\mathbb{Z}/N\mathbb{Z})^*$, вычисляет сообщение $m = (c^d \bmod N)$.

Проверим согласованность этой схемы. Действительно, $\mathrm{Dec}(sk, \mathrm{Enc}(pk, m)) \equiv (m^e)^d = m^{ed} \equiv m \pmod{N}$ по теореме Эйлера ($m^{\varphi(N)} \equiv 1 \pmod{N}$, если $\gcd(m, N) = 1$).

Надежность этой схемы связана с задачей о разложении числа на простые множители: если известны простые p, q, для которых $N = pq$, то по открытому ключу (N, e) можно восстановить и закрытый ключ $d \equiv e^{-1} \pmod{(p-1)(q-1)}$. Обратно, оказывается, что по числу N вида $N = pq$ и числам e, d таким, что $ed \equiv 1 \pmod{\varphi(N)}$, можно вычислить множители числа N за полиномиальное время. Конечно, возможно, что сообщение m можно восстановить из шифра c и другим способом, не вычисляя множителей p и q.

Как мы говорили выше, описанная схема не надежна ни в каком из рассматриваемых нами смыслов — хотя бы потому, что она детерминирована.

3.3.2. Практические вопросы. Схема RSA умеет зашифровывать элементы из $(\mathbb{Z}/N\mathbb{Z})^*$. Пусть l — длина числа N в битах. Тогда каждую строку m длины $l-1$ можно истолковать как элемент $\mathbb{Z}/N\mathbb{Z}$. Но что, если этот элемент окажется необратимым? Во-первых, вероятность этого (при случайном выборе m) чрезвычайно мала (порядка 2^{-n}). Во-вторых, даже если хакер захочет найти такое m специально, ему вряд ли это удастся: дело в том, что если $1 < m < N$ и $\gcd(m, N) \neq 1$, то этот наибольший общий делитель является нетривиальным делителем N, то есть, тогда известно разложение N на простые множители! В-третьих, даже если m — необратимый элемент $\mathbb{Z}/N\mathbb{Z}$, то алгоритм дешифровки все равно восстанавливает его из $c = m^e$.

Действительно, покажем, что $m^{ed} \equiv m \pmod{N}$ при *всех* m. Мы не можем воспользоваться теоремой Эйлера, но можем вспомнить малую теорему Ферма (то есть, частный

случай теоремы Эйлера для простого модуля) и применить ее отдельно по модулю p и модулю q. Напомним, что $ed \equiv 1 \pmod{(p-1)(q-1)}$. Если $m \equiv 0 \pmod{p}$, то сравнение $m^{ed} \equiv m \pmod{p}$ выполнено по тривиальным причинам. Если же $m \not\equiv 0 \pmod{p}$, то можно применить малую теорему Ферма: $m^{p-1} \equiv 1 \pmod{p}$, и потому $m^{ed} \equiv m \pmod{p}$. Аналогично, $m^{ed} \equiv m \pmod{q}$. Но тогда из китайской теоремы об остатках следует, что $m^{ed} \equiv m \pmod{pq}$, что и требовалось.

Наконец, получатель может использовать китайскую теорему об остатках для ускорения дешифровки. Для вычисления c^d по модулю N достаточно вычислить c^d по модулям p и q (которые известны получателю). Но $c^d \equiv c^{d \bmod (p-1)} \pmod{p}$ и $c^d \equiv c^{d \bmod (q-1)} \pmod{q}$. Показатели $d \bmod (p-1)$ и $d \bmod (q-1)$ никак не зависят от c, и их можно вычислить заранее, а возводить в такие степени проще, чем в степень d.

Для упрощения операции зашифровки на практике часто выбираются не очень большие значения e, например, $e = 3$. Однако, нужно помнить, что наивная схема RSA очень плохо ведет себя при таком выборе e. Действительно, если $m < N^{1/3}$, то при вычислении $c = m^3 \bmod N$ не происходит редукции по модулю N, и потому полученное значение c является точным кубом. Разумеется, из целого числа совсем несложно извлечь кубический корень. Ситуация $m < N^{1/3}$ вовсе не гипотетическая: представьте, что мы используем гибридную схему шифрования и выбрали в качестве закрытого ключа k 128-битную случайную строку. Тогда при зашифровке k с помощью алгоритма RSA с параметром $n = 512$ бит и $e = 3$ произойдет описанная выше неприятность: k^3 имеет длину, меньшую длины N (которая равна примерно $2n$).

3.3.3. RSA с набивкой. Как мы говорили выше, наивная схема RSA не имеет шансов быть надежной в смысле любого из наших определений в силу своей детерминированности. Поэтому мы обязаны сделать шифрование рандомизированным. Простейший способ сделать это — приписать к сообщению перед зашифровкой случайную строку. Полученная схема называется **RSA с набивкой**.

Генерация ключа в ней выглядит так же, как и в наивном варианте RSA. Пусть l — некоторая функция такая, что $l(n) \leqslant 2n - 2$. Для зашифровки сообщения $m \in \{0,1\}^{l(n)}$ выберем случайным образом строку r (она называется *набивкой*) из $\{0,1\}^{||N||-l(n)-1}$ (условие $l(n) \leqslant 2n-2$ гарантирует, что строка r непуста). Тогда длина конкатенации $r||m$ равна $||N|| - 1$, и поэтому можно воспринимать $r||m$ как элемент $\mathbb{Z}/N\mathbb{Z}$. Зашифруем его как в наивном RSA, то есть положим $c = m^e \bmod N$. Для расшифровки нужно, как и в наивном RSA, возвести c в степень d по модулю N, получив $r||m$, и взять последние $l(n)$ бит результата.

Является ли эта схема надежной? Ответ зависит от того, как ведет себя функция $l(n)$: если $l(n)$ слишком мала, то «случайности» r не хватает для надежности. В то же время, если $l(n)$ слишком велика, то на сообщение остается мало бит, и схема становится неэффективной. Кроме того, как всегда, ответ зависит от того, к каким предположениям мы сводим этот вопрос. Опишем стандартную задачу, в сложность которой мы верим. Она просто утверждает, что мы не умеем эффективно извлекать корень степени e по модулю N.

Пусть $\mathcal{A}$ — некоторый хакер, GenRSA — алгоритм генерации ключа RSA (см. раздел 3.3.1). Эксперимент RSA-inv$_{\mathcal{A},\mathrm{GenRSA}}(n)$ выглядит следующим образом:

1. Мы запускаем $\mathrm{GenRSA}(n)$ и получаем (N, e, d).
2. Выбираем случайным образом $y \in (\mathbb{Z}/N\mathbb{Z})^*$ и отправляем хакеру N, e, y.
3. Хакер выводит $x \in (\mathbb{Z}/N\mathbb{Z})^*$.

4. Результат эксперимента равен 1 (успех), если $x^e = y \pmod N$.

Определение 3.3.3.1. Будем говорить, что **задача RSA сложна относительно** GenRSA, если для любого полиномиального хакера существует пренебрежимо малая функция ε такая, что $P(\text{RSA-inv}_{\mathcal{A},\text{GenRSA}}(n) = 1) \leqslant \varepsilon(n)$.

Вернемся к вопросу о надежности схемы RSA с набивкой. Если функция l слишком большая, так что $l(n) = 2n - O(\ln n)$, то длина набивки r равна $O(\ln n)$. Поэтому мы можем перебрать все возможные значения r за полиномиальное время $2^{O(\ln n)}$. Если пространство возможных сообщений m не очень большое, это дает хакеру возможность взлома нашей схемы полным перебором.

Естественно предположить, что при $l(n) = c \cdot n$ схема надежна, но у нас нет доказательства этого, основанного на предположении сложности задачи RSA. Однако, имеются доказательства надежности такой схемы, основанные на нестандартных (более сильных) предположениях.

Наконец, если $l(n)$ не очень большая, то схема RSA с набивкой надежна, как показывает следующая теорема.

Теорема 3.3.3.2. *Если задача RSA сложна по отношению к* GenRSA*, то схема RSA с набивкой, в которой* $l(n) = O(\ln n)$*, устойчива относительно chosen plaintext-атак.*

В одном из популярных криптографических стандартов PKCS #1 используется RSA с набивкой следующим образом. Предположим, что сообщение m имеет длину, кратную 8, то есть, состоит из целого числа байтов. Выберем случайную строку r, не содержащую нулевых байтов, после чего рассмотрим строку $(00000000||00000010||r||00000000||m)$ и проделаем с ней операцию шифрования, как в наивном RSA. За счет выбора r без нулевых байтов при расшифровке значение m востанавливается однозначно без предварительного знания длины m.

3.3.4. Схема Эль-Гамаля. Пусть G — конечная группа, и $m \in G$ — произвольный элемент. Если g — равномерно распределенный случайный элемент группы G, то произведение $g' = m \cdot g$ тоже равномерно распределено. Таким образом, g' не несет никакой информации об исходном элементе m. Можно представить схему шифрования с закрытым ключом, в которой сообщение m при зашифровке умножается на ключ g: $c = m \cdot g$. На самом деле, схема одноразовых блокнотов ровно в этом и состоит: строки длины n образуют абелеву группу относительно операции побитового сложения $\oplus$.

Для шифрования с открытым ключом мы не можем брать истинно случайный элемент g, но можно взять *псевдослучайный*; знание некоторой информации об этом элементе (некоторого секретного ключа) позволяет быструю расшифровку.

Пусть $\mathcal{G}$ — некоторый алгоритм генерации групп (см. раздел 3.1.3). Схема шифрования Эль-Гамаля $S = (\text{Gen}, \text{Enc}, \text{Dec})$ устроена следующим образом:

- Алгоритм генерации ключа Gen принимает на вход натуральное число n и запускает $\mathcal{G}(n)$, получая группу G порядка q с образующей g. После этого он выбирает случайным образом x из $\mathbb{Z}/q\mathbb{Z}$ и вычисляет $h = g^x$. Открытый ключ — это набор (G, q, g, h), а секретный ключ — набор (G, q, g, x).
- Алгоритм шифрования Enc получает открытый ключ $pk = (G, q, g, h)$ и сообщение $m \in G$. Он выбирает случайным образом y из $\mathbb{Z}/q\mathbb{Z}$ и выводит $c = (g^y, h^y \cdot m)$.
- Алгоритм дешифрования Dec по секретному ключу $sk = (G, q, g, x)$ и зашифрованному сообщению $c = (c_1, c_2)$ выводит $m = c_2/c_1^x$.

Несложно проверить согласованность этой схемы: если $(c_1, c_2) = (g^y, h^y \cdot m) = (g^y, g^{xy} \cdot m)$, то $c_2/c_1^x = g^{xy} \cdot m/(g^y)^x = m$.

Теорема 3.3.4.1. *Если задача DDH сложна относительно алгоритма генерации групп $\mathcal{G}$, то схема шифрования Эль Гамаля надежна относительно chosen plaintext-атак.*

Доказательство. Достаточно доказать, что схема S сложна относительно подслушивания. Пусть $\mathcal{A}$ — полиномиальный хакер. Положим

$$\varepsilon(n) = P(\mathrm{PubK}^{eav}_{\mathcal{A},S}(n) = 1).$$

Рассмотрим модифицированную «схему шифрования» $\widetilde{S}$, в которой алгоритм Gen такой же, как в S, а алгоритм шифрования Enc по открытому ключу $pk = (G, q, g, h)$ и сообщению m выбирает случайным образом элементы y, z из $\mathbb{Z}/q\mathbb{Z}$ и выводит пару $c = (g^y, g^z \cdot m)$. Заметим, что такое сообщение невозможно расшифровать, так что $\widetilde{S}$ вообще не является схемой шифрования. Тем не менее, ничто не мешает нам использовать ее в эксперименте по взлому $\mathrm{PubK}^{eav}_{\mathcal{A},\widetilde{S}}(n)$. Первая компонента зашифрованного сообщения c равномерна распределена по G по очевидным причинам. Вторая компонента тоже равномерно распределена (это мы обсуждали в начале раздела); это распределение никак не зависит от сообщения m. Поэтому $P(\mathrm{PubK}^{eav}_{\mathcal{A},\widetilde{S}}(n) = 1) = 1/2$.

Для сведения надежности схемы S к сложности задаче DDH предъявим полиномиальный алгоритм D, который, опираясь на хакера $\mathcal{A}$, пытается решить задачу DDH. Напомним, что на вход D подается набор (G, q, g, g_1, g_2, g_3), где $g_1 = g^x$, $g_2 = g^y$, и $g_3 = g^{xy}$ или g^z для независимых равномерно распределенных $x, y, z \in \mathbb{Z}/q\mathbb{Z}$.

Итак, алгоритм D получает G, q, g, g_1, g_2, g_3 и действует следующим образом.

1. Пусть $pk = (G, q, g, g_1)$; запустим $\mathcal{A}(pk)$ и получим два сообщения m_0, m_1.
2. Выберем случайный бит b и положим $c_1 = g_2$, $c_2 = g_3 \cdot m_b$.
3. Отправим хакеру $\mathcal{A}$ шифр (c_1, c_2) и получим от него бит b'. Выведем 1, если $b' = b$, и 0 в противоположном случае.

Что происходит при работе алгоритма D? Предположим сначала, что он получил на вход набор (G, q, g, g^x, g^y, g^z) для случайных $x, y, x \in \mathbb{Z}/q\mathbb{Z}$. Тогда он отдает хакеру $\mathcal{A}$ открытый ключ (G, q, g, g^x) и зашифрованное сообщение $(c_1, c_2) = (g^y, g^z \cdot m_b)$. Таким образом, с точки зрения $\mathcal{A}$ происходит «издевательский» эксперимент $\mathrm{PubK}^{eav}_{\mathcal{A},\widetilde{S}}(n)$. Кроме того, D выводит 1 в точности в случае $b' = b$, то есть, в случае успеха этого эксперимента. Как мы показали выше, вероятность этого равна $1/2$.

Второй случай: алгоритм D получил на вход набор $(G, q, g, g^x, g^y, g^{xy})$ для случайных $x, y \in \mathbb{Z}/q\mathbb{Z}$. Тогда он отдает хакеру $\mathcal{A}$ открытый ключ (G, q, g, g^x) и зашифрованное сообщение $(c_1, c_2) = (g^y, g^{xy} \cdot m_b)$. Значит, с точки зрения хакера $\mathcal{A}$ он проходит эксперимент $\mathrm{PubK}^{eav}_{\mathcal{A},S}(n)$. Снова D выводит 1 в точности в случае $b = b'$, то есть, в случае успеха этого эксперимента. По предположению, вероятность этого равна $\varepsilon(n)$. Поэтому

$$|P(D(G, q, g, g^x, g^y, g^z) = 1) - P(D(G, q, g, g^x, g^y, g^{xy}) = 1)| = |\frac{1}{2} - \varepsilon(n)|.$$

С другой стороны, по предположению сложности задачи DDH, эта разность является пренебрежимо малой функцией. Поэтому $\varepsilon(n)$ отличается от $1/2$ на пренебрежимо малую функцию, что и требовалось доказать. □

3.4 Дальнейшие схемы шифрования

3.4.1. Квадратичные вычеты по простому модулю. Посмотрим на остатки по простому нечетному модулю p. Элемент $x \in (\mathbb{Z}/p\mathbb{Z})^*$ называется **квадратичным вычетом** по модулю p, если он является квадратом некоторого обратимого остатка, то есть, если $x = y^2$ для некоторого $y \in (\mathbb{Z}/p\mathbb{Z})^*$. В противном случае он называется **квадратичным невычетом**.

Заметим, что отображение возведения в квадрат $(\mathbb{Z}/p\mathbb{Z})^* \to (\mathbb{Z}/p\mathbb{Z})^*$, $y \mapsto y^2$, «склеивает» по два элемента; то есть, у каждого обратимого остатка либо два квадратных корня, либо ни одного. Действительно, если $x^2 \equiv x'^2 \pmod p$, то $(x - x')(x + x') \equiv 0 \pmod p$, поэтому либо $x - x'$, либо $x + x'$ делится на p. Это и означает, что либо $x \equiv x' \pmod p$, либо $x \equiv -x' \pmod p$ (заметим, что $-x' \not\equiv x' \pmod p$ в силу нечетности p). Из этого немедленно следует, что ровно половина обратимых остатков по модулю p являются квадратичными вычетами, и потому квадратичных вычетов и квадратичных невычетов поровну: по $(p-1)/2$. Мы будем обозначать множество всех квадратичных вычетов по модулю p через QR_p, а множество всех квадратичных невычетов — через QNR_p.

Пусть g — некоторый *первообразный корень* по модулю p, то есть, элемент, порождающий [циклическую] группу $(\mathbb{Z}/p\mathbb{Z})^*$. Таким образом, $g^{p-1} \equiv 1 \pmod p$ и $g^k \not\equiv 1 \pmod p$ при $1 \leqslant k \leqslant p-2$. Тогда любой элемент группы $(\mathbb{Z}/p\mathbb{Z})^*$ можно записать в виде g^a для единственного целого a такого, что $0 \leqslant a \leqslant p-2$. Иными словами, $g^0, g^1, \ldots, g^{p-2}$ — все элементы $(\mathbb{Z}/p\mathbb{Z})^*$. Заметим, что элементы вида g^{2k} являются квадратичными вычетами: $g^{2k} = (g^k)^2$. Таких элементов уже $(p-1)/2$ (при $k = 0, 1, \ldots, (p-3)/2$), поэтому все остальные элементы — элементы вида g^{2k+1} — должны быть квадратичными невычетами.

Умеем ли мы эффективно определять, является ли остаток по модулю p квадратичным вычетом или невычетом? Да, умеем. Для этого определим **символ Якоби** остатка $x \in (\mathbb{Z}/p\mathbb{Z})^*$ следующим образом:

$$J_p(x) = \begin{cases} 1, & \text{если } x \in \mathrm{QR}_p; \\ -1, & \text{если } x \in \mathrm{QNR}_p\,. \end{cases}$$

Теорема 3.4.1.1. $J_p(x) \equiv x^{(p-1)/2} \pmod p$.

Доказательство. Выберем первообразный корень g по модулю p. Если x — квадратичный вычет, то $x = g^{2k}$, и потому $x^{(p-1)/2} = (g^{2k})^{(p-1)/2} = g^{k(p-1)} = (g^{p-1})^k = 1$ (по малой теореме Ферма). Если же $x = g^{2k+1}$, то есть, $x \in \mathrm{QNR}_p$, то $x^{(p-1)/2} = (g^{2k+1})^{(p-1)/2} = g^{k(p-1)} \cdot g^{(p-1)/2} = g^{(p-1)/2}$. Чему же равно $g^{(p-1)/2}$? Заметим, что $(g^{(p-1)/2})^2 = g^{p-1} = 1$, поэтому $g^{(p-1)/2}$ должен быть квадратным корнем из 1. Мы знаем, что таких корня ровно два: 1 и -1. Но $g^{(p-1)/2} \neq 1$ в силу того, что g — *первообразный* корень по модулю p. □

Следствие 3.4.1.2. *Символ Якоби по модулю* p мультипликативен*:* $J_p(xy) = J_p(x) \cdot J_p(y)$ *для* $x, y \in (\mathbb{Z}/p\mathbb{Z})^*$.

Доказательство. $J_p(xy) = (xy)^{(p-1)/2} = x^{(p-1)/2} \cdot y^{(p-1)/2} = J_p(x) \cdot J_p(y)$. □

Мультипликативность можно сформулировать следующим образом: если x, y одновременно являются квадратичными вычетами или одновременно являются квадратичными невычетами, то произведение xy — квадратичный вычет. Если же один из элементов x, y вычет, а второй невычет, то их произведение xy — невычет.

3.4.2. Квадратичные вычеты по составному модулю. Перейдем теперь к изучении ситуации по модулю $N = pq$, где p, q — различные нечетные простые числа. Китайская теорема об остатках говорит нам, что кольца $\mathbb{Z}/N\mathbb{Z}$ и $(\mathbb{Z}/p\mathbb{Z}) \times (\mathbb{Z}/q\mathbb{Z})$ изоморфны. Поэтому и группы обратимых элементов в них изоморфны:

$$(\mathbb{Z}/N\mathbb{Z})^* \cong (\mathbb{Z}/p\mathbb{Z})^* \times (\mathbb{Z}/q\mathbb{Z})^*.$$

Мы постоянно будем использовать этот изоморфизм: если $x \in (\mathbb{Z}/N\mathbb{Z})^*$, мы будем писать $x \leftrightarrow (x_p, x_q)$ — здесь $x_p = x \bmod p$, $x_q = x \bmod q$. Обратно, для любой пары $a \in (\mathbb{Z}/p\mathbb{Z})^*$, $b \in (\mathbb{Z}/q\mathbb{Z})^*$ однозначно определен элемент $x \in (\mathbb{Z}/N\mathbb{Z})^*$ такой, что $x \leftrightarrow (a, b)$.

Как и раньше, назовем элемент $x \in (\mathbb{Z}/N\mathbb{Z})^*$ **квадратичным вычетом** по модулю N, если он является квадратом некоторого элемента из $(\mathbb{Z}/N\mathbb{Z})^*$, и **квадратичным невычетом** в противном случае. Что происходит с вычетами и невычетами при описанном изоморфизме? Оказывается, x является квадратичным вычетом по модулю N тогда и только тогда, когда x_p — квадратичный вычет по модулю p, а x_q — квадратичный вычет по модулю q. Действительно, если $x = y^2$, то $x_p = y_p^2$ и $x_q = y_q^2$. Обратно, если $x_p = a^2$ и $x_q = b^2$ для некоторых $a \in (\mathbb{Z}/p\mathbb{Z})^*$, $b \in (\mathbb{Z}/p\mathbb{Z})^*$, то элемент $y \leftrightarrow (a, b)$ при возведении в квадрат дает x.

Это означает, что квадратичных вычетов по модулю N теперь ровно $1/4$ от всех обратимых остатков по модулю N: для получения квадратичного вычета из пары остатков $(a, b) \in (\mathbb{Z}/p\mathbb{Z})^* \times (\mathbb{Z}/q\mathbb{Z})^*$ нужно, чтобы и a, и b был квадратичным вычетом, а таких пар ровно

$$\frac{p-1}{2} \cdot \frac{q-1}{2} = \frac{(p-1)(q-1)}{4} = \frac{\varphi(N)}{4}.$$

Множество всех квадратичных вычетов по модулю N обозначим через QR_N, а множество всех квадратичных невычетов — через QNR_N.

Отображение возведения в квадрат $(\mathbb{Z}/N\mathbb{Z})^* \to (\mathbb{Z}/N\mathbb{Z})^*$ теперь склеивает по четыре элемента в один: пара (x_p, x_q) имеет тот же квадрат, что и пары $(x_p, -x_q)$, $(-x_p, x_q)$, $(-x_p, -x_q)$.

Символ Якоби по модулю N определяется как произведение символов Якоби по модулям p и q:

$$J_N(x) = J_p(x) \cdot J_q(x).$$

При таком определении мультипликативность символа Якоби очевидна.

Предложение 3.4.2.1. *Символ Якоби по модулю* N мультипликативен*:* $J_N(xy) = J_N(x) \cdot J_N(y)$ *для* $x, y \in (\mathbb{Z}/N\mathbb{Z})^*$.

Доказательство. $J_N(xy) = J_p(xy)J_q(xy) = J_p(x)J_p(y)J_q(x)J_q(y) = J_N(x)J_N(y)$. □

Поскольку J_p принимает значение 1 для половины обратимых остатков по модулю p, значение -1 для другой половины, и то же самое можно сказать про J_q, то $J_N(x)$ равно 1 для половины остатков из $(\mathbb{Z}/N\mathbb{Z})^*$ и -1 для другой половины. Выше мы видели, что x является квадратичным вычетом тогда и только тогда, когда $J_p(x) = J_q(x) = 1$. Значит, $J_N(x) = 1$ для квадратичного вычета x. С другой стороны, есть и квадратичные невычеты, для которых $J_N(x) = 1$: это остатки, для которых $J_p(x) = J_q(x) = -1$. Множество таких квадратичных невычетов по модулю N мы обозначим через QNR_N^+.

Итак, все обратимые остатки по модулю N разбиваются на две половины: остатки с $J_N(x) = 1$ и остатки с $J_n(x) = -1$. Далее, остатки с $J_N(x) = 1$ разбиваются еще на две равные части: QR_N и QNR_N^+. В частности, $|\,QR_N\,| = |\,QNR_N^+\,|$.

Мы могли бы попробовать построить схему шифрования, основанную на том, что сложно отличить квадратичный вычет от квадратичного вычета по модулю N, не зная разложения N на простые множители. Однако, существует эффективный алгоритм, вычисляющий $J_N(x)$, даже если разложение N неизвестно. Таким образом, у нас есть частичный тест на определение квадратичных вычетов: если $J_N(x) = -1$, то x точно является квадратичным невычетом. Вместе с тем, если все же $J_N(x) = 1$, то x может оказаться и квадратичном вычетом ($x \in QR_N$), и квадратичным невычетом ($x \in QNR_N^+$). Для различения этих групп уже неизвестно ни одного эффективного алгоритма.

Определение 3.4.2.2. Задача определения квадратичных вычетов сложна по отношению к GenModulus, если для любого полиномиального алгоритма $\mathcal{A}$ существует пренебрежимо малая функция ε такая, что

$$|P(\mathcal{A}(N, qr) = 1) - P(\mathcal{A}(N, qnr) = 1)| \leqslant \varepsilon(n),$$

где вероятности берутся по выводам GenModulus(n) и по случайно выбранным qr из QR_N, qnr из QNR_N^{+1}.

3.4.3. Схема шифрования Гольдвассер–Микали. На сложности отличения квадратичных вычетов от квадратичных невычетов основана следующая **схема шифрования Гольдвассер–Микали**:

- Алгоритм генерации ключей Gen, получив параметр n, запускает алгоритм GenModulus(n) и получает (N, p, q), где $N = pq$, и p, q — нечетные простые числа длины n. Алгоритм выводит открытый ключ N и закрытый ключ (p, q).
- Алгоритм зашифровки Enc принимает открытый ключ N и сообщение $m \in \{0, 1\}$. Если $m = 0$, он выводит случайно выбранный квадратичный вычет по модулю N (то есть, элемент QR_N), а если $m = 1$, то случайно выбранный квадратичный невычет по модулю N с символом Якоби 1 (то есть, элемент QNR_N^+). Таким образом, $c \in (\mathbb{Z}/N\mathbb{Z})^*$.
- Алгоритм дешифровки Dec, получив закрытый ключ (p, q) и зашифрованное сообщение $c \in (\mathbb{Z}/N\mathbb{Z})^*$, вычисляет $J_p(c)$ и $J_q(c)$. Если $J_p(c) = J_q(c) = 1$, он выводит $m = 0$, а если $J_p(c) = J_q(c) = -1$, он выводит $m = 1$.

Нам осталось пояснить, как алгоритм зашифровки генерирует случайные элементы QR_N и QNR_N^+. Получить квадратичный вычет несложно: нужно взять случайный элемент $x \in (\mathbb{Z}/N\mathbb{Z})^*$, и тогда x^2 окажется случайным элементом QR_N (распределение останется равномерным, поскольку возведение в квадрат склеивает по четыре остатка).

Чуть сложнее получить случайный элемент QNR_N^+, но получатель, генерирующий ключи, может в этом помочь. А именно, получатель может сгенерировать один элемент $z \in QNR_N^+$ и включить его в открытый ключ. После этого для получения случайного элемента QNR_N^+ достаточно выбрать случайный элемент $y \in QR_N$ (как выше) и вычислить произведение $z \cdot y$. Проверим, что оно лежит в QNR_N^+: действительно, $J_p(z) = J_q(z) = -1$ и $J_p(y) = J_q(y) = 1$. В силу мультипликативности символа Якоби получаем, что $J_p(zy) = J_q(zy) = -1$, и потому $zy \in QNR_N^+$. Оказывается, знание элемента $z \in QNR_N^+$ не уменьшит надежность этой схемы.

Отметим, что генерация $z \in QNR_N^+$ при известных p, q не очень сложна: можно брать случайные элементы $z \in (\mathbb{Z}/N\mathbb{Z})^*$, пока не попадется такой, для которого $J_p(z) = J_q(z) = -1$. Есть, однако, ненулевая вероятность, что нам не повезет, но она экспоненциально убывает с ростом числа попыток, и потому дает лишь пренебрежимо малую вероятность ошибки.

Таким образом, зашифровку можно описать так: для зашифровки бита $m \in \{0,1\}$ мы выбираем случайный элемент $x \in (\mathbb{Z}/N\mathbb{Z})^*$ и выводим зашифрованное сообщение $c = z^m \cdot x^2 \bmod N$.

Теорема 3.4.3.1. *Если задача определения квадратичных вычетов сложна по отношению к* GenModulus*, то схема шифрования Гольдвассер–Микали устойчива относительно chosen plaintext-атак.*

3.4.4. Извлечение квадратных корней. Следующая схема шифрования (схема Рабина) основана на сложности вычисления квадратного корня в кольце вычетов. Обсудим сначала, как извлекать квадратные корни по простому модулю.

Пусть $y \in (\mathbb{Z}/p\mathbb{Z})^*$. Для существования обратимого остатка x такого, что $x^2 = y$, необходимо и достаточно, чтобы y было квадратичным вычетом по модулю p, что, как мы знаем, может быть эффективным образом определено. Если мы уже знаем, что $J_p(y) = 1$, как вычислить такой x?

Это несложно сделать, если p дает остаток 3 при делении на 4: $p = 4k + 3$. В этом случае по теореме 3.4.1.1 выполнено $y^{(p-1)/2} \equiv 1 \pmod p$, и $(p-1)/2$ — нечетное число. Домножая обе части на y, получаем $y^{(p+1)/2} \equiv y \pmod p$. Теперь правая часть — это $y^{(4k+4)/2} = y^{2k+2} = (y^{k+1})^2$, поэтому один квадратный корень из y — это y^{k+1}, а второй, конечно, $-y^{k+1}$.

Пусть теперь p дает остаток 1 при делении на 4: $p = 4k + 1$. Мы утверждаем, что тогда $y^r c^{r'} \equiv 1 \pmod p$ для некоторых показателей r, r' таких, что r нечетно, r' четно, а c — некоторый (на самом деле, произвольный фиксированный) квадратичный невычет по модулю p. Если мы этого добьемся, то можно повторить трюк из рассмотрения предыдущего случая: тогда $y^{r+1}c^{r'} \equiv y$, и потому $(y^{(r+1)/2}c^{r'/2})^2 \equiv y \pmod p$.

Пока что мы знаем лишь, что $y^{(p-1)/2}c^0 \equiv 1 \pmod p$, где $(p-1)/2$ четно. Для того, чтобы его превратить в нечетное число, мы будем делить его на 2, одновременно подправляя показатель у c. А именно, пусть $(p-1)/2 = 2^l \cdot m$, где m нечетно (числа l и m эффективно вычисляются последовательным делением на 2). Опишем первый шаг: $y^{2^{l-1}\cdot m} \cdot c^0 \equiv 1 \pmod p$, то есть, $(y^{2^{l-1}\cdot m} \cdot c^0)^2 \equiv 1 \pmod p$. Значит, выражение в скобках равно 1 или -1; если оно равно 1 — все хорошо, мы уменьшили степень двойки в показателе элемента y. Если же оно оказалось равно -1: $y^{2^{l-1}\cdot m} \cdot c^0 \equiv -1 \pmod p$. Но, поскольку c — квадратичный невычет, то $c^{2^l\cdot m} = c^{(p-1)/2} \equiv -1 \pmod p$. Перемножая эти два сравнения, получаем, что $y^{2^{l-1}\cdot m} \cdot c^{2^l\cdot m} \equiv 1 \pmod p$. У нас снова получилось выражение вида $y^r c^{r'} \equiv 1 \pmod p$ с четным r'; если r уже нечетно, мы победили, а если четно ($l > 1$), можно продолжить процесс.

Более формально: пусть $y^r \cdot c^{r'} = 1$, где $r = (p-1)/2 = 2^l \cdot m$, $r' = 0$. Последовательно для $i = l, \ldots, 1$ будем заменять $r \mapsto r/2$, $r' \mapsto r'/2$; и если обнаружится, что $a^r \cdot c^{r'} = -1$, то подправим $r' \mapsto r' + 2^l \cdot m$. Нетрудно понять, что в конце мы придем к ситуации, когда $r = m$ нечетно, r' четно, и все еще $y^r \cdot c^{r'} = 1$ — этого мы и добивались.

Отметим, что для применения этого алгоритма мы должны выбрать какой-нибудь квадратичный невычет b, так что мы получили рандомизированный алгоритм; науке неизвестно, существует ли детерминированный алгоритм для вычисления квадратного корня по простому модулю.

Теперь понятно, как извлекать корень по модулю $N = pq$, если p и q известны: для $y \in (\mathbb{Z}/N\mathbb{Z})^*$ мы находим $a \in (\mathbb{Z}/p\mathbb{Z})^*$, $b \in (\mathbb{Z}/q\mathbb{Z})^*$ такие, что $a^2 \equiv y_p \pmod p$, $b^2 \equiv y_q \pmod q$. Тогда элемент $x \leftrightarrow (a, b)$ будет давать в квадрате y (более того, можно найти и остальные три квадратных корня из y).

3.4.5. Схема шифрования Рабина. Опишем эксперимент по вычислению корня $\mathrm{SQR}_{\mathcal{A},\mathrm{GenModulus}}(n)$.

1. Мы генерируем (N, p, q) при помощи $\mathrm{GenModulus}(n)$, выбираем случайно $z \in (\mathbb{Z}/N\mathbb{Z})^*$, вычисляем $y = z^2 \bmod N$ и отправляем хакеру $\mathcal{A}$ пару (N, y) (таким образом, y — случайный элемент множества QR_N).
2. Хакер выводит остаток $x \in (\mathbb{Z}/N\mathbb{Z})^*$.
3. Результат эксперимента равен 1, если $x^2 = y$, и 0 в противном случае.

Определение 3.4.5.1. Мы говорим, что **задача вычисления квадратных корней сложна** по отношению к GenModulus, если для любого полиномиального хакера $\mathcal{A}$ существует пренебрежимо малая функция ε такая, что $\mathrm{P}(\mathrm{SQR}_{\mathcal{A},\mathrm{GenModulus}}(n) = 1) \leqslant \varepsilon(n)$.

Выше мы заметили, что если эта задача сложна, то и задача разложения на простые сложна. Оказывается, верно и обратное.

Лемма 3.4.5.2. *Пусть* $N = pq$, p, q — *различные нечетные простые. Пусть даны* x, x' *такие, что* $x^2 = x'^2 \pmod N$, *но* $x \neq \pm x' \pmod N$. *Тогда можно разложить* N *за полиномиальное (по* $||N||$*) время.*

Доказательство. Пусть $x \leftrightarrow (x_p, x_q)$. По нашему условию тогда $x' \leftrightarrow (-x_p, x_q)$ или $x' \leftrightarrow (x_p, -x_q)$. В первом случае $x + x' \leftrightarrow (0, 2x_q)$; то есть, $x + x'$ делится на p, но не делится на q. Поэтому $\gcd(x + x', N) = p$. Во втором случае, аналогичным образом, $\gcd(x + x', N) = q$. □

Теорема 3.4.5.3. *Если задача разложения на простые сложна по отношению к* GenModulus, *то и задача вычисления квадратных корней сложна по отношению к* GenModulus.

Опишем схему шифрования Рабина, основанную на задаче вычисления квадратичных корней. Для этого нам понадобится алгоритм GenModulus, который возвращает тройку (N, p, q) такую, что $N = pq$, числа p и q простые и имеют длину n, а также выполнено дополнительное условие: $p \equiv q \equiv 3 \pmod 4$.

1. Алгоритм генерации ключей Gen, получив параметр n, запускает $\mathrm{GenModulus}(n)$ и получает тройку (N, p, q), удовлетворяющую условию, описанному выше. Открытый ключ — число N, закрытый ключ — пара (p, q).
2. Алгоритм шифрования Enc, получив N и однобитовое сообщение $m \in \{0, 1\}$, выбирает случайным образом x из QR_N и выводит $c = ((x^2 \bmod N), \mathrm{lsb}(x) \oplus m)$, где $\mathrm{lsb}(x)$ — последний бит в двоичной записи числа x.
3. Алгоритм дешифровки Dec, получив p, q и $c = (c_1, c_2)$, извлекает все четыре квадратных корня из c_1 и выбирает единственный из них, который является квадратичным вычетом по модулю N. Пусть это x. Тогда $m = \mathrm{lsb}(x) \oplus c_2$.

Покажем, что при расшифровке в схеме шифрования Рабина действительно ровно один из четырех возможных корней из c_1 является квадратичным вычетом. Заметим, что $c_1 = x^2 \bmod N$, а четыре квадратичных корня из c_1 равны $(\pm x_p, \pm x_q)$. Нетрудно видеть, что -1 является квадратичным невычетом по модулю p и по модулю q: $(-1)^{(p-1)/2} = -1$, поскольку $p \equiv 3 \pmod 4$. По мультипликативности символа Якоби, когда мы меняем знак у x_p или у x_q, соответствующий символ Якоби меняет знак. Поэтому из четырех приведенных пар ровно в одной оба символа Якоби равны 1, что равносильно тому, что этой паре соответствует квадратичный вычет по модулю N.

Мы не будем доказывать надежность схемы Рабина — она, разумеется, основана на еще одном недоказанном предположении: $\mathrm{lsb}(x)$ сложно вычислить, зная лишь $x^2 \bmod N$ и тот факт, что $x \in \mathrm{QR}_N$.

3.4.6. Остатки по модулю N^2. Для описания схемы шифрования Пайе нам понадобится знание группы обратимых остатков по модулю N^2.

Предложение 3.4.6.1. *Пусть* $N = pq$, *где* p, q — *различные нечетные простые числа одной длины.*

1. $\gcd(N, \varphi(N)) = 1$.
2. $(1+N)^a \equiv 1 + aN \pmod{N^2}$ *для всех* $a \geqslant 0$. *Как следствие, элемент* $1+N$ *имеет порядок* N *в* $(\mathbb{Z}/N^2\mathbb{Z})^*$.
3. *Группа* $(\mathbb{Z}/N^2\mathbb{Z})^*$ *изоморфна* $(\mathbb{Z}/N\mathbb{Z}) \times (\mathbb{Z}/N\mathbb{Z})^*$. *При этом изоморфизме пара* $(a, b) \in (\mathbb{Z}/N\mathbb{Z}) \times (\mathbb{Z}/N\mathbb{Z})^*$ *переходит в* $(1+N)^a \cdot b^N \in (\mathbb{Z}/N^2\mathbb{Z})^*$.

Доказательство. Мы докажем первые два пункта. У нас $N = pq$ и $\varphi(N) = (p-1)(q-1)$. Не умаляя общности, можно считать, что $p > q$. Если $\gcd(pq, (p-1)(q-1)) > 1$, то он равен p или q (он не может равняться N, поскольку $\varphi(N) < N$). Он не может равняться p: тогда $(p-1)(q-1)$ делится на p, и потому $p-1$ или $q-1$ делится на p, но оба этих числа меньше p. Если же $\gcd(pq, (p-1)(q-1)) = q$, то $(p-1)(q-1)$ делится на q, и потому $p-1$ делится на q. Очевидно, что $p - 1 \neq q$, потому $(p-1)/q \geqslant 2$, откуда $p - 1 > 2q$, что невозможно, поскольку p и q имеют одну длину в битах.

Второе утверждение сразу следует из разложения $(1+N)^a$ по биному Ньютона. □

Обратите внимание, что изоморфизм в третьем пункте предложения эффективно вычислим, даже если не знать разложения N на простые множители (а вот обратный к нему — нет). Далее мы будем обозначать этот изоморфизм так: $x \leftrightarrow (a, b)$, где $x = (1+N)^a \cdot b^N \in (\mathbb{Z}/N^2\mathbb{Z})^*$. Не забывайте, что в этой записи a берется из аддитивной группы $\mathbb{Z}/N\mathbb{Z}$, а b — из мультипликативной группы $(\mathbb{Z}/N\mathbb{Z})^*$. Поэтому $(a, b) \cdot (a', b') = (a + a', bb')$.

Будем говорить, что элемент $y \in (\mathbb{Z}/N^2\mathbb{Z})^*$ является **N-вычетом по модулю N^2**, если y является N-й степенью, то есть, существует $x \in (\mathbb{Z}/N^2\mathbb{Z})^*$ такой, что $y = x^N \pmod{N^2}$. Множество всех N-вычетов по модулю N^2 мы будем обозначать через $\mathrm{Res}(N^2)$. Пусть y — некоторый N-вычет. Тогда $y = x^N$ для некоторого $x \leftrightarrow (a, b)$, и потому $y \leftrightarrow (Na, b^N) = (0, b^N)$. Это значит, что все N-вычеты имеют вид $(0, *)$. С другой стороны, отображение $b \mapsto b^N$ в группе $(\mathbb{Z}/N\mathbb{Z})^*$ сюръективно, поскольку N взаимно просто с порядком этой группы $\varphi(N)$. Стало быть, N-вычеты по модулю N^2 — это в точности элементы вида $(0, c)$, $c \in (\mathbb{Z}/N\mathbb{Z})^*$. Также понятно, что отображение $x \mapsto x^N$ на $(\mathbb{Z}/N^2\mathbb{Z})^*$ склеивает по N элементов в один.

3.4.7. Схема шифрования Пайе. Эта схема основана на том предположении, что невозможно эффективно отличить N-вычет по модулю N^2 от произвольного элемента $(\mathbb{Z}/N^2\mathbb{Z})^*$ (если не знать разложение N на простые множители).

Определение 3.4.7.1. Будем говорить, что **задача определения составного вычета сложна** по отношению к GenModulus, если для любого полиномиального хакера $\mathcal{A}$ существует пренебрежимо малая функция ε такая, что

$$|P(\mathcal{A}(N, r^N \bmod N^2) = 1) - P(\mathcal{A}(N, r) = 1)| \leqslant \varepsilon(n),$$

где вероятности берутся по всем выводам (N, p, q) алгоритма GenModulus и по всем случайным равномерно распределенным $r \in (\mathbb{Z}/N^2\mathbb{Z})^*$ (таким образом, $r^N \bmod N^2$ — случайный равномерно распределенный элемент $\mathrm{Res}(N^2)$).

Идея схемы Пайе состоит в следующем: пусть N — открытый ключ. Для сообщения $m \in \mathbb{Z}/N\mathbb{Z}$ выберем случайный N-вычет $(0, r)$ и положим $c = (m, 1) \cdot (0, r) = (m, r)$. Если задача определения составного вычета сложна, то $(0, r)$ неотличим от (r', r) для случайных r', r, а потому c неотличим от $(m, 1) \cdot (r', r) = (m + r', r)$. При этом $m + r'$ равномерно распределен по $\mathbb{Z}/N\mathbb{Z}$, и потому распределение пары $(m + r', r)$ не зависит от m. Более подробно:

1. Алгоритм генерации ключей Gen получает параметр n, запускает GenModulus(n) и получает (N, p, q). Открытый ключ — число N, закрытый ключ — пара (p, q).

2. Алгоритм шифрования Enc получает открытый ключ N и сообщение $m \in \mathbb{Z}/N\mathbb{Z}$. Он выбирает случайным образом $r \in (\mathbb{Z}/N\mathbb{Z})^*$ и выводит $c = (1 + N)^m \cdot r^N \bmod N^2$.

3. Алгоритм дешифровки Dec получает $c \in (\mathbb{Z}/N^2\mathbb{Z})^*$, вычисляет $c' = c^{\varphi(N)} \bmod N^2$, $m' = (c' - 1)/N$, и выводит $m = (m' \cdot \varphi(N)^{-1} \bmod N)$.

Проверим согласованность схемы Пайе: если $c = (1 + N)^m \cdot r^N$, то

$$\begin{aligned} c' &\equiv c^{\varphi(N)} \\ &\equiv (1 + N)^{m\varphi(N)} \cdot r^{N\varphi(N)} \\ &\equiv (1 + N)^{m\varphi(N)} \cdot (r^{\varphi(N)})^N \\ &\equiv (1 + N)^{m\varphi(N)} \\ &\equiv 1 + m\varphi(N)N \pmod{N^2} \end{aligned}$$

Поэтому $c' - 1 \equiv m\varphi(N)N \pmod{N^2}$. После деления на N получаем, что $m' = (c' - 1)/N \equiv m\varphi(N) \pmod{N}$. Домножая на обратный остаток к $\varphi(N)$ по модулю N (такой существует, поскольку $\gcd(N, \varphi(N)) = 1$), получаем исходное сообщение m.

Теорема 3.4.7.2. *Если задача определения составного вычета сложна по отношению к* GenModulus*, то схема шифрования Пайе устойчива относительно chosen plaintext-атак.*

Определение 3.4.7.3. Схема шифрования с открытым ключом $S = (\text{Gen}, \text{Enc}, \text{Dec})$ называется **гомоморфной**, если для всех n и для всех пар (pk, sk), выводимых Gen(n), можно определить группы M, C такие, что Enc$(pk, -)$ зашифровывает сообщения из M, и все результаты лежат в C, и для любых $m_1, m_2 \in M$, $c_1, c_2 \in C$ таких, что $m_1 = \text{Dec}(sk, c_1)$, $m_2 = \text{Dec}(sk, c_2)$, выполнено $\text{Dec}_{sk}(c_1 \cdot c_2) = m_1 \cdot m_2$ (где операции выполняются в группах M, C).

Несложное вычисление показывает, что схема шифрования Пайе является гомоморфной: $(1 + N)^{m_1} r_1^N \cdot (1 + N)^{m_2} r_2^N = (1 + N)^{m_1 + m_2} (r_1 r_2)^N$. Упражнение: покажите, что схема шифрования Эль-Гамаля также является гомоморфной.

В качестве примера применения гомоморфного шифрования приведем схему голосования. Пусть l голосующих хотят проголосовать по какому-то вопросу; у каждого имеется голос «да» или «нет».

1. Авторитетная инстанция выбирает и публикует открытый ключ N для схемы шифрования Пайе.

2. Пусть 0 означает «нет», 1 означает «да». Голосующие зашифровывают свои голоса: если v_i — голос, то $c_i = ((1+N)^{v_i} \cdot r^N \bmod N^2)$ — зашифрованный голос, где r случайно выбирается из $(\mathbb{Z}/N\mathbb{Z})^*$.

3. Голосующие обмениваются своими зашифрованными голосами c_i и вычисляют $c^* = \prod_{i=1}^{l} c_i \bmod N^2$.

4. Инстанция получает c^*, расшифровывает и получает $v^* = \sum_{i=1}^{l} v_i \mod N$; равенство выполнено без модуля, если l достаточно мало.

Таким образом, инстанция получает правильный результат голосования, не узнав ни одного индивидуального голоса. Более того, никто не узнает чужой голос, и результат голосования поддается проверке.

3.5 Цифровая подпись

3.5.1. Общая схема. Цифровая подпись позволяет обладателю секретного ключа «подписывать» сообщения таким образом, что любой обладатель открытого ключа может проверить, что именно это сообщение было действительно подписано владельцем секретного ключа. Отметим, что в такой схеме один ключ работает для всех получателей сообщений (в отличии от кодов аутентификации сообщений, где необходимо генерировать свой закрытый ключ для каждого получателя). Более того, если один из них проверил подпись на данном сообщении, то он уверен в том, что и у остальных получателей эта проверка приведет к тому же результату. Поэтому получатель сообщения с цифровой подписью может убедить и третью сторону в том, что сообщение подписано отправителем. Кроме того, отправитель не может позднее отказаться от своей подписи.

Определение 3.5.1.1. Тройка вероятностных полиномиальных алгоритмов $(\mathrm{Gen}, \mathrm{Sign}, \mathrm{Vrfy})$ называется **схемой подписи**, если выполнены следующие условия.

1. **Алгоритм генерации ключа** Gen по параметру n выдает пару ключей (pk, sk), состоящую из **открытого ключа** pk и **закрытого ключа** sk. Мы предполагаем, что длина pk и sk не меньше n.

2. **Алгоритм подписи** Sign по закрытому ключу sk и сообщению $m \in \{0,1\}^*$ выводит подпись $\sigma \leftarrow \mathrm{Sign}(sk, m)$.

3. **Алгоритм проверки** Vrfy по открытому ключу pk, сообщению m и подписи σ выводит бит b; значение $b = 1$ соответствует результату «корректно», а значение $b = 0$ — результату «некорректно». Обозначение: $b = \mathrm{Vrfy}(pk, m, \sigma)$.

4. Условие согласованности: для всех n, для всех пар (pk, sk), которые могут быть выведены алгоритмом $\mathrm{Gen}(n)$, для всех $m \in \{0,1\}^*$ выполнено

$$\mathrm{Vrfy}(pk, m, \mathrm{Sign}(sk, m)) = 1$$

5. Мы, как обычно, предполагаем, что алгоритм проверки Vrfy является детерминистским.

6. Алгоритм $\mathrm{Sign}(sk, -)$ может быть определен лишь для сообщений длины $l(n)$, зависящей от n — тогда мы говорим о **схеме подписи для сообщений длины** $l(n)$.

7. В конкретных реализациях иногда полезно допустить, что условие согласованности выполняется с вероятностью, отличающейся от 1 на пренебрежимо малую величину.

Пусть P сгенерировал открытый ключ pk. Будем говорить, что хакер $\mathcal{A}$ **сгенерировал подделку**, если ему удалось сгенерировать сообщение m и подпись σ такие, что $\mathrm{Vrfy}(\mathrm{pk}, m, \sigma) = 1$, но сообщение m ранее не подписывалось P. Схема подписи считается надежной, если хакер не может сгенерировать подделку, даже если он может получать подписи произвольных сообщений.

Пусть $S = (\mathrm{Gen}, \mathrm{Sign}, \mathrm{Vrfy})$ — схема подписи. Определим эксперимент по подделке подписи $\mathrm{SigForge}_{\mathcal{A},S}(n)$ для хакера $\mathcal{A}$ и параметра n.

1. Мы запускаем $\mathrm{Gen}(n)$ и получаем ключи $(\mathrm{pk}, \mathrm{sk})$.
2. Хакер $\mathcal{A}$ получает ключ pk и доступ к оракулу $\mathrm{Sign}(\mathrm{sk}, -)$. Хакер выводит пару (m, σ). Пусть Q — множество сообщений, подпись к которым хакер запрашивал в процессе своей работы.
3. Результат эксперимента равен 1, если $\mathrm{Vrfy}(\mathrm{pk}, m, \sigma) = 1$ и $m \notin Q$, и 0 в противном случае.

Определение 3.5.1.2. Схема подписи S называется **экзистенциально неподделываемой**, если для любого полиномиального хакера $\mathcal{A}$ существует пренебрежимо малая функция ε такая, что $P(\mathrm{SigForge}_{\mathcal{A},S}(n) = 1) \leqslant \varepsilon(n)$.

3.5.2. Наивный RSA. Приведем простой пример цифровой подписи, не являющийся надежным в смысле определения 3.5.1.2. Он чрезвычайно похож на наивную схему шифрования RSA (раздел 3.3.1).

- Алгоритм генерации ключа Gen принимает на вход параметр n и запускает $\mathrm{GenModulus}(n)$. Получив тройку (N, e, d) такую, что $N = pq$ для простых чисел p, q длины n, $ed \equiv 1 \pmod{\varphi(n)}$, он возвращает открытый ключ (N, e) и закрытый ключ (N, d).
- Алгоритм подписи Sign по закрытому ключу (N, d) и сообщению $m \in (\mathbb{Z}/N\mathbb{Z})^*$ генерирует подпись $\sigma = m^d \bmod N$.
- Алгоритм проверки Vrfy по открытому ключу (N, e), сообщению m и подписи σ вычисляет σ^e и сравнивает его с m; в случае равенства он выводит 1, а в противном случае — 0.

Обратите внимание, что описанная схема умеет подписывать сообщения из $(\mathbb{Z}/N\mathbb{Z})^*$, а не битовые строки, как того требует схема 3.5.1.1. Подписи, сгенерированные этой схемой, проходят проверку по той же причине, по которой схема шифрования RSA является согласованной.

Конечно, описанная схема не является надежной: хакер в эксперименте SigForge может получить открытый ключ $\mathrm{pk} = (N, e)$ и вывести корректную пару (m, σ), не задав ни одного вопроса оракулу: достаточно выбрать произвольным образом $\sigma \in (\mathbb{Z}/N\mathbb{Z})^*$ и вычислить $\sigma^e \bmod N$. Конечно, он никак не контролирует полученное сообщение m, но с точки зрения определения 3.5.1.2 это неважно.

Возможно получить и подпись к наперед заданному сообщению $m \in (\mathbb{Z}/N\mathbb{Z})^*$: выберем произвольное m_1 и вычислим $m_2 = m/m_1 \bmod N$. Попросим оракула подписать m_1 и m_2, получив подписи σ_1 и σ_2. Тогда $\sigma = \sigma_1\sigma_2$ будет корректной подписью для m. Действительно, $\sigma^e = (\sigma_1\sigma_2)^e = \sigma_1^e\sigma_2^e = m_1m_2 = m$.

3.5.3. RSA с хэшем. Сейчас мы модифицируем наивную схему подписи RSA и получим что-то более разумное. Идея в том, что перед зашифровкой методом RSA нужно *хэшировать* сообщение. А именно, пусть $H\colon \{0,1\}^* \to (\mathbb{Z}/N\mathbb{Z})^*$ — некоторая функция, определение которой мы включим в открытый ключ. Мы будем подписывать сообщение m подписью $\sigma = H(m)^d \bmod N$. Для проверки корректности пары (m, σ) достаточно сравнить σ^e с $H(m)$.

Сразу отметим, что функция H должна быть односторонней — то есть, по $H(m)$ должно быть сложно восстановить m. Иначе против этой схемы работает фактически та же атака, что и против наивной подписи RSA: возьмем произвольное σ, вычислим σ^e и вычислим m такое, что $H(m) = \sigma^e$.

Кроме того, H должна быть *устойчивой к коллизиям*. Это означает, что должно быть сложно предъявить пару сообщений m_1, m_2 такую, что $H(m_1) = H(m_2)$. Действительно, при наличии такой пары достаточно попросить оракула подписать сообщение m_1, получить подпись σ, и тогда пара $((m_2, \sigma))$ также пройдет проверку на корректность.

Более общо, можно взять любую схему цифровой подписи для сообщений ограниченной длины, скажем, длины $l(n) = n$, взять набор хэш-функций, предоставляющий функции вида $H\colon \{0,1\}^* \to \{0,1\}^n$, и получить схему цифровой подписи для сообщений произвольной длины. Эта схема берет сообщение m, вычисляет $H(m)$ и подписывает $H(m)$ при помощи цифровой подписи для сообщений ограниченной длины. Можно показать, что если исходная схема экзистенциально неподделываема, то и полученная также экзистенциально неподделываема.

3.5.4. Схема одноразовой подписи Лэмпорта. Сейчас мы определим схему, которая «надежна» только когда она используется для подписи одного сообщения. Определим для этого эксперимент по одноразовой подделке подписи $\mathrm{SigForge}^1_{\mathcal{A},S}(n)$.

1. Мы запускаем $\mathrm{Gen}(n)$ и получаем ключи (pk, sk).
2. Мы отправляем хакеру открытый ключ pk. Он просит оракула $\mathrm{Sign}(sk, -)$ подписать *одно* сообщение m'. После этого хакер выводит пару (m, σ) с $m \neq m'$.
3. Результат эксперимента равен 1, если $\mathrm{Vrfy}(pk, m, \sigma) = 1$, и 0 в противном случае.

Определение 3.5.4.1. Схема цифровой подписи S называется **одноразовой экзистенциально неподделываемой**, если для любого полиномиального хакера $\mathcal{A}$ существует пренебрежимо малая функция ε такая, что $P(\mathrm{SigForge}^1_{\mathcal{A},S}(n) = 1) \leqslant \varepsilon(n)$.

Схема Лэмпорта будет зашифровывать сообщения длины $l = l(n)$ следующим образом:

- Алгоритм генерации ключа Gen получает n и выбирает случайным образом $2l$ элементов $(x_{i,0}, x_{i,1})_{1 \leqslant i \leqslant l}$ из $\{0,1\}^n$. После этого он вычисляет $y_{i,0} = f(x_{i,0})$ и $y_{i,1} = f(x_{i,1})$. Открытый ключ pk состоит из всех значений $(y_{i,0}, y_{i,1})$, а закрытый ключ sk — из значений $(x_{i,0}, x_{i,1})$.
- Алгоритм генерации подписи по закрытому ключу sk и сообщению $m =\in \{0,1\}^l$ выводит подпись $(x_{1,m[1]}, \ldots, x_{l,m[l]})$.
- Алгоритм проверки получает открытый ключ, сообщение $m \in \{0,1\}^l$ и подпись $\sigma = (x_1, \ldots, x_l)$. Он выводит 1 тогда и только тогда, когда выполнены равенства $f(x_i) = y_{i,m[i]}$ для всех $1 \leqslant i \leqslant l$.

Оказывается, если f — односторонняя функция, то описанная схема подписи является одноразовой экзистенциально неподделываемой. Действительно, знание подписи для сообщения m' дает хакеру знание значений $x_{i,m'[i]}$ для всех i. Сообщение m должно отличаться от m' хотя бы в одном бите i, и для корректной подписи хакер фактически должен найти $x_{i,m[i]}$ по $y_{i,m[i]}$ для этого i. Но эта задача равносильна задаче обращения одного значения функции f.

3.6 Другие криптографические протоколы

3.6.1. Сертификаты. До сих пор мы обсуждали лишь, как использовать криптографию с открытым ключом, если открытые ключи для передачи сообщений уже известны. Например, в описании схем шифрования мы говорили, что если Алиса хочет отправить Бобу сообщение, то она может зашифровать его с помощью открытого ключа Боба. Но откуда она получит этот ключ так, чтобы быть уверенной, что он действительно принадлежит Бобу? Она не может воспользоваться для этого открытыми каналами связи, так как сообщения может перехватить хакер и отправить ей свой открытый ключ, выдав себя за Боба. Сейчас мы покажем, как использовать криптографию с открытым ключом для надежного распространения открытых ключей.

Ключом здесь является понятие **цифрового сертификата**. Неформально говоря, это подпись, удостоверяющая, что некоторый открытый ключ принадлежит некоторому лицу. Предположим, что Чарли сгенерировал пару ключей (pk_C, sk_C), и у Боба есть пара ключей (pk_B, sk_B). Предположим, что Чарли знает, что pk_B действительно является открытым ключом Боба. Тогда Чарли может сформировать сообщение вида «Ключ Боба — это pk_B», подписать его своим открытым ключом и отдать результат Бобу. Результат

$$cert_{C\to B} = \mathrm{Sign}(sk_C, \text{«ключ Боба — это } pk_B\text{»})$$

называется **сертификатом** для ключа Боба, выпущенным Чарли.

Теперь предположим, что Боб хочет, чтобы Алиса послала ему сообщение. Если Алиса уже знает открытый ключ pk_C Чарли, Боб может послать ей пару $(pk_B, cert_{C\to B})$. Алиса теперь может проверить, что $cert_{C\to B}$ действительно является подписью сообщения «ключ Боба — это pk_B». Если Алиса доверяет Чарли, то она может быть уверена, что pk_B действительно является открытым ключом Боба. Отметим, что при этом все сообщения между Алисой и Бобом могут посылаться по незащищенному каналу связи.

Осталось понять, как Алиса может с самого начала узнать ключ pk_C (и почему она вообще доверяет Чарли), и как Чарли может быть уверен в том, что pk_B действительно является открытым ключом Боба. Одним из решений является наличие *сертифицирующего авторитета* — организации, которая занимается выдачей сертификатов пользователям. Например, это может быть отдел в компании, созданный для того, чтобы сотрудники могли общаться друг с другом. Для сертификации публичного ключа Боба тогда Боб должен войти в непосредственный контакт с этой организацией — например, прийти в организацию и предъявить свое удостоверение личности.

Однако, если пользователи сертифицирующей организации не ограничены пределами одной компании, возникают проблемы: вряд ли все вокруг готовы доверять одной организации. Тогда можно положиться на несколько независимых сертифицирующих авторитетов: Боб может получить сертификаты своего открытого ключа от нескольких организаций, а Алиса может выбирать, каким организациям она доверяет.

Еще один способ — цепочки сертификатов. Вернемся к нашему примеру, в котором Чарли выдал Бобу сертификат $cert_{C\to B}$, удостоверяющий, что pk_B — открытый ключ Боба. После этого Боб, узнав открытый ключ Алисы pk_A, может выдать ей сертификат

$\mathrm{cert}_{B\to A}$, удостоверяющий, что pk_A действительно является открытым ключом Алисы. Теперь для распространения своего открытого ключа Алиса может использовать набор $(\mathrm{pk}_A, \mathrm{cert}_{B\to A}, \mathrm{pk}_B, \mathrm{cert}_{C\to B})$. Этот набор удостоверяет, что Чарли (которому, предположительно, все доверяют) выдал сертификат, из которого следует, что pk_B — открытый ключ Боба, а этот самый Боб подписал сертификат о том, что pk_A является открытым ключом Алисы. Если мы верим тому, что Чарли выдает сертификаты только надежным людям (и, более того, людям, которым он доверяет выдачу других сертификатов), то из этой цепочки можно заключить, что pk_A действительно является открытым ключом Алисы.

Наконец, для сертифицирования ключей шифрования электронной почты PGP используется полностью распределенная система: любой пользователь может выдать сертификат любому другому, и каждый может решать, насколько он доверяет каждому из других.

Часто полезно ограничить срок действия сертификата: для этого достаточно прписать к подписываемому сообщению об открытом ключе дату истечения этого срока. Для проверки такого сертификата нужно знать теперь не только открытый ключ, но и эту дату, и проверять, что она еще не прошла.

В случае утечки закрытого ключа (или, например, если сотрудник покидает организацию) полезно иметь возможность немедленного отзыва сертификата (а не дожидаться, пока истечет срок его действия). Для этого можно, например, добавить к сообщению о сертифицировании открытого ключа серийный номер. После этого сертифицирующая организация может публиковать ежедневно на своем сайте серийные номера отозванных сертификатов (разумеется, подписав такой список своим ключом). Проверка такого сертификата не ограничивается корректностью подписи, а должна включать проверку, находится ли его серийный номер в соответствующем списке на сайте.

3.6.2. Доказательства с нулевым разглашением. Предположим, что Алиса хочет убедить Боба в истинности некоторого факта, не предоставляя ему слишком много информации. А именно, она хочет, чтобы Боб не смог убедить других людей в том, что этот факт истинен. Протокол, который обеспечивает это, называется *доказательством с нулевым разглашением*.

Как правило, такой протокол состоит в нескольких раундах обмена информацией между Алисой и Бобом, после которого Боб проверяет корректность полученной информации, и в зависимости от результата объявляет раунд успешным или неуспешным. Этот обмен должен быть организован так, что если доказываемый факт истинен, то *каждый* раунд должен завершиться успехом. Если же он ложен, то вероятность того, что каждый раунд завершится успехом, должна быть пренебрежимо малой величиной (как правило, экспоненциально убывать в зависимости от количества раундов).

Приведем конкретный пример такого протокола. Пусть Алиса знает разложение числа $N = pq$ на два различных простых множителя p и q. Предположим, что она хочет убедить Боба в том, что некоторый остаток $y \in (\mathbb{Z}/N\mathbb{Z})^*$ является квадратичным вычетом. При этом Бобу известно N, но неизвестны p и q (иначе он сам мог бы проверить, является ли y вычетом). Алиса, зная p и q, может вычислить некоторый квадратный корень x из y, то есть, ей известен и остаток $x \in (\mathbb{Z}/N\mathbb{Z})^*$ такой, что $x^2 \equiv y \pmod N$.

Каждый раунд обмена информацией состоит в следующем:

1. Алиса выбирает случайный остаток r по модулю N. Она вычисляет $s = r^2 \bmod N$ и посылает результат Бобу.

2. Боб выбирает случайный бит b из $\{0, 1\}$ и посылает его Алисе.

3. Алиса посылает Бобу число

$$z = \begin{cases} r, & \text{если } b = 0; \\ xr, & \text{если } b = 1. \end{cases}$$

4. Боб вычисляет $z^2 \bmod N$ и проверяет, что

$$z^2 \equiv \begin{cases} s \pmod N, & \text{если } b = 0; \\ ys \pmod N, & \text{если } b = 1. \end{cases}$$

Боб объявляет раунд успешным, если сравнение выполнено, и неуспешным в противном случае. Так повторяется большое количество (скажем, $n = 80$) раз. Нетрудно проверить, что если y является квадратичным вычетом, то $z \equiv x^b r \pmod N$, и потому $z^2 \equiv x^{2b} r^2 \equiv y^b s \pmod N$, а значит, Боб будет объявлять каждый раунд успешным.

Предположим теперь, что y не является квадратичным вычетом. Тогда (при любом выборе s) только одно из значений s, ys является квадратичным вычетом по модулю N. Поэтому в каждом раунде с вероятностью $1/2$ Алиса не сможет в принципе ответить на вопрос Боба: ей понадобится извлечь квадратный корень из s или из ys. Поэтому вероятность правильного ответа Алисы на n раундов не превосходит 2^{-n}.

Покажем теперь, что результат общения Боба с Алисой никак не поможет ему убедить кого-нибудь третьего в том, что y действительно является квадратичным вычетом. Все, что осталось у Боба после разговора с Алисой — это набор троек $(s_1, b_1, z_1), (s_2, b_2, z_2), \ldots$, где $z_i^2 \equiv y^{b_i} s_i \pmod N$. Но сейчас мы покажем, что Боб может создать такой набор и без помощи Алисы. Действительно, в исходном эксперименте все, что зависит от Боба — это выбор бита $b = 0$ или $b = 1$. Теперь представим, что Боб выбирает бит $b \in \{0, 1\}$ и остаток $z \in (\mathbb{Z}/N\mathbb{Z})^*$. Тогда он может вычислить $s \equiv z^2 (y^b)^{-1} \pmod N$, и тройка (s, b, z) будет удовлетворять тому же сравнению, что и любая из троек (s_i, b_i, z_i). Поэтому с помощью списка, полученного при разговоре с Алисой, Боб не может убедить никого в том, что y является квадратичным вычетом: если бы он мог это сделать, то он мог бы сделать это и с помощью своего сгенерированного списка без помощи Алисы (даже если y не является квадратичным вычетом).

3.6.3. Разделение секрета. Схема разделения секрета решает, как нетрудно догадаться, задачу разделения секрета между несколькими людьми. Допустим, нам нужно, чтобы банковская ячейка открывалась только в присутствии троих людей. Тогда можно выдать каждому из них по одной трети от длинной комбинации цифр, открывающей сейф. Проблема здесь в том, что если двое из трех соберутся вместе и решат открыть сейф без третьего, им нужно будет перебрать гораздо меньше вариантов, чем тому, кто не знает ни одной цифры из комбинации. Хотелось бы, чтобы в схеме разделения секрета между n людьми никакая группа из $n - 1$ из них не имела бы преимущества перед остальными взломщиками.

Такую схему несложно предложить: для того, чтобы засекретить остаток $S \in \mathbb{Z}/m\mathbb{Z}$ и распределить секрет между n людьми, выберем случайным образом $n-1$ остаток $D_1, D_2, \ldots, D_{n-1} \in \mathbb{Z}/m\mathbb{Z}$ и положим $D_n = S - D_1 - \cdots - D_{n-1} \bmod m$. После этого i-й участник получает D_i, и, собравшись вместе, они могут посчитать сумму $S = D_1 + \cdots + D_n$. В то же время, никакие $n - 1$ из них ничего не знают про S.

Более общая задача: мы хотим распределить секрет между n людьми так, чтобы любые t из них могли восстановить секрет, но никакие $t - 1$ из них не могли. Первые схемы

разделения секреты для $t < n$ были построены независимо друг от друга Ади Шамиром и Джорджем Блэкли в 1979 году.

Схема разделения секрета Шамира основана на том, что многочлен степени не выше k полностью определяется своими значениями в $k + 1$ точке. Для разделения секрета S положим $a_0 = S$ и выберем случайным образом числа $a_1, \ldots, a_{t-1}$. Рассмотрим многочлен $f = a_0 + a_1 x + a_2 x^2 + \cdots + a_{t-1} x^{t-1}$ Выберем случайным образом n точек $x_1, \ldots, x_n$ и вычислим $y_i = f(x_i)$ для всех $i = 1, \ldots, n$ (например, можно взять просто $x_i = i$). Выдадим i-му участнику значение y_i. Теперь, если мы знаем t из пар (x_i, y_i), мы можем восстановить по ним единственный многочлен степени не выше $t - 1$, который в точке x_i равен y_i — это и есть многочлен f.