☰ Оглавление ▼

Иллюстрация работы RSA на примере

Вокруг алгоритмов шифрования с отрытым и закрытым ключом существует множество недопониманий и мистификаций. Здесь я хотел бы предельно коротко и наглядно, с конкретными числами и минимумом формул, показать, как это работает.

Я не вдаюсь в теорию (не очень понятно, на какой уровень подготовки читателя следует рассчитывать), но я уверен, что прочитав эту короткую иллюстрацию, любому человеку будет проще разобраться в формулах и строгих доказательствах.

Итак. Допустим, я хочу получить от вас некие данные. Мы с вам не хотим, чтобы эти данные узнал кто-то, кроме нас. И у нас нет никакой уверенности в надёжности канала передачи данных. Приступим.

Шаг первый. Подготовка ключей

Я должен проделать предварительные действия: сгенерировать публичный и приватный ключ.

- Выбираю два простых числа. Пусть это будет $p=3$ и $q=7$.
- Вычисляем *модуль* — произведение наших p и q: $n=p\times q=3\times 7=21$.
- Вычисляем *функцию Эйлера*: $\phi=(p-1)\times(q-1)=2\times 6=12$.
- Выбираем число e, отвечающее следующим критериям: (i) оно должно быть простое, (ii) оно должно быть меньше ϕ — остаются варианты: 3, 5, 7, 11, (iii) оно должно быть взаимно простое с ϕ; остаются варианты 5, 7, 11. Выберем $e=5$. Это, так называемая, *открытая экспонента*.

Теперь пара чисел $\{e, n\}$ — это мой открытый ключ. Я отправляю его вам, чтобы вы зашифровали своё сообщение. Но для меня это ещё не всё. Я должен получить закрытый ключ.

Мне нужно вычислить число d, обратное e по модулю ϕ. То есть остаток от деления по модулю ϕ произведения $d\times e$ должен быть равен 1. Запишем это в обозначениях, принятых во многих языках программирования: $(d\times e)\%\phi=1$. Или $(d\times 5)\%12=1$. d может быть равно 5 ($(5\times 5)\%12=25\%12=1$), но чтобы оно не путалось с e в дальнейшем повествовании, давайте возьмём его равным 17. Можете проверить сами, что $(17\times 5)\%12$ действительно равно 1 ($17\times 5-12\times 7=1$). Итак $d=17$. Пара $\{d, n\}$ — это секретный ключ, его я оставляю у себя. Его нельзя сообщать никому. Только обладатель секретного ключа может расшифровать то, что было зашифровано открытым ключом.

Шаг второй. Шифрование

Теперь пришла ваша очередь шифровать ваше сообщение. Допустим, ваше сообщение это число 19. Обозначим его $P=19$. Кроме него у вас уже есть мой открытый ключ: $\{e, n\} = \{5, 21\}$. Шифрование выполняется по следующему алгоритму:

- Возводите ваше сообщение в степень e по модулю n. То есть, вычисляете 19 в степени 5 (2476099) и берёте остаток от деления на 21. Получается 10 — это ваши закодированные данные.

Строго говоря, вам вовсе незачем вычислять огромное число «19 в степени 5». При каждом умножении достаточно вычислять не полное произведение, а только остаток от деления на 21. Но это уже детали реализации вычислений, давайте не будем в них углубляться.

Полученные данные `E=10`, вы отправляете мне.

Здесь надо заметить, что сообщение `P=19` не должно быть больше `n=21`. иначе ничего не получится.

Шаг третий. Расшифровка

Я получил ваши данные (`E=10`), и у меня имеется закрытый ключ `{d, n} = {17, 21}`.

Обратите внимание на то, что открытый ключ не может расшифровать сообщение. А закрытый ключ я никому не говорил. В этом вся прелесть асимметричного шифрования.

Начинаем раскодировать:

- Я делаю операцию, очень похожую на вашу, но вместо `e` использую `d`. Возвожу `E` в степень `d`: получаю 10 в степени 17 (позвольте, я не буду писать единичку с семнадцатью нулями). Вычисляю остаток от деления на 21 и получаю 19 — ваше сообщение.

Заметьте, никто, кроме меня (даже вы!) не может расшифровать ваше сообщение (`E=10`), так как ни у кого нет закрытого ключа.

В чём гарантия надёжности шифрования

Надёжность шифрования обеспечивается тем, что третьему лицу (старающемуся взломать шифр) очень трудно вычислить закрытый ключ по открытому. Оба ключа вычисляются из одной пары простых чисел (`p` и `q`). То есть ключи связаны между собой. Но установить эту связь очень сложно. Основной загвоздкой является декомпозиция модуля `n` на простые сомножители `p` и `q`. Если число является произведением двух очень больших простых чисел, то его очень трудно разложить на множители.

Постараюсь это показать на примере. Давайте разложим на множители число 360:

- сразу ясно, что оно делится на два (получили 2)
- оставшееся 180 тоже, очевидно чётное (ещё 2)
- 90 — тоже чётное (ещё двойка)
- 45 не делится на 2, но следующая же попытка оказывается успешной — оно делится на три (получили 3)
- 15 тоже делится на 3
- 5 — простое.

Мы на каждом шагу, практически без перебора, получали всё новые и новые множители, легко получив полное разложение 360=2×2×2×3×3×5

Давайте теперь возьмём число 361. Тут нам придётся помучиться.

- оно не чётное
- три — нет, не делится
- пять (допустим, мы поступаем умно и перебираем только простые числа, хотя, на практике, поиск больших простых чисел, сам по себе, является сложной задачей) — не подходит
- семь? — нет.
- …
- и только 19 даст нам ответ: 361=19×19.

При использовании больших чисел, задача становится очень сложной. Это позволяет надеяться, что у взломщика просто не хватит вычислительных ресурсов, чтобы сломать ваш

шифр за обозримое время.

А как это всё работает на практике?

Многие читатели спрашивают, как всё это применяется на практике. Давайте рассмотрим чуть более приближенный к жизни пример. Зашифруем и расшифруем слово «КРОТ», предложенное одним из читателей. А заодно, бегло рассмотрим, какие проблемы при этом встречаются и как они решаются.

Сперва сгенерируем ключи с чуть бóльшими числами. Они не так наглядны, но позволят нам шифровать не только числа от нуля до 20.

Оттолкнёмся от пары простых чисел `{p, q} = {17, 19}`. Пусть наш открытый ключ будет `{e, n} = {5, 323}`, а закрытый `{d, n} = {173, 323}`.

Мы готовы к шифрованию. Переведём наше слово в цифровое представление. Мы можем взять просто номера букв в алфавите. У нас получится последовательность чисел: 11, 17, 15, 19.

Мы можем зашифровать каждое из этих чисел открытым ключом `{e, n} = {5, 323}` и получить шифровку 197, 272, 2, 304. Эти числа можно передать получателю, обладающему закрытым ключом `{d, n} = {173, 323}` и он всё расшифрует.

Немного о сложностях

На самом деле, изложенный способ шифрования очень слаб и никогда не используется. Причина проста — шифрование по буквам. Одна и та же буква будет шифроваться одним и тем же числом. Если злоумышленник перехватит достаточно большое сообщение, он сможет догадаться о его содержимом. Сперва он обратит внимание на частые коды пробелов и разделит шифровку на слова. Потом он заметит однобуквенные слова и догадается, как кодируются буквы «а», «и», «о», «в», «к»… Путём недолгого перебора, он вычислит дополнительные буквы по коротким словам, типа «но», «не», «по». И по более длинным словам без труда восстановит все оставшиеся буквы.

Таким образом, злоумышленнику не придётся отгадывать ваши секретные ключи. Он взломает ваше сообщение, не зная их.

Чтобы этого не происходило, используются специальные дополнительные алгоритмы, суть которых в том, что каждая предыдущая часть сообщения начинает влиять на следующую.

Упрощённо, это выглядит так. Перед шифрованием, мы применяем к сообщению правило: `b := (b + a) % n`. Где `a` — предыдущая часть сообщения, а `b` — следующая. То есть наше сообщение (11, 17, 15, 19) изменяется. 11 остаётся без изменений. 17 превращается в `(11 + 17) % 323 = 28`. 15 становится `(15 + 28) % 323 = 43`. А 19 превращается в 62.

Последовательность (11, 28, 43, 62) получается «запутанной». Все буквы в ней как бы перемешаны, в том смысле, что на каждый код влияет не одна буква, а все предыдущие.

Тот, кто получит ваше сообщение, должен будет проделать обратную операцию, со знаком «минус»: `b := (b - a) % n`. И только тогда он получит коды букв.

На практике, в исходное сообщение специально добавляются случайные и бессмысленные буквы в начало. Чтобы даже по первому коду было невозможно ничего понять. Получатель просто отбрасывает начало сообщения.

То есть мы можем добавить случайное число в начало и получить (299, 11, 17, 15, 19). После перемешивания получится: 299, 310, 4, 19, 38. После шифрования уже невозможно будет догадаться где была какая буква.

В реальной жизни всё ещё немного сложнее. Блоки, на которые бьётся сообщение длиннее одной буквы. Поэтому, сперва применяются алгоритмы выравнивания, потом алгоритмы

разбиения на блоки с перепутыванием, и только потом применяется само RSA-шифрование.

Получатель делает всё в обратном порядке: расшифровывает, «распутывает» блоки и отбрасывает ненужную информацию, добавленную просто для выравнивания (чтобы сообщение можно было разбить на целое число блоков).

Детали и принципы формирования блоков можно почитать тут. Я же в этой заметке хотел рассказать только про RSA. Надесь, удалось.
