Алгоритмические проблемы теории чисел

Ю. В. Нестеренко

Эта статья посвящена алгоритмам теории чисел. Вопрос «как сосчитать?» всегда сопутствовал теоретико-числовым исследованиям. Труды Евклида и Диофанта, Ферма и Эйлера, Гаусса, Чебышева и Эрмита содержат остроумные и весьма эффективные алгоритмы решения диофантовых уравнений, выяснения разрешимости сравнений, построения больших по тем временам простых чисел, нахождения наилучших приближений и т.д. Без преувеличения можно сказать, что вся теория чисел пронизана алгоритмами. В последние два десятилетия, благодаря в первую очередь запросам криптографии и широкому распространению ЭВМ, исследования по алгоритмическим вопросам теории чисел переживают период бурного и весьма плодотворного развития. Мы кратко затронем здесь лишь те алгоритмические аспекты теории чисел, которые связаны с криптографическими применениями. За рамками статьи останутся проблемы нахождения решений диофантовых уравнений, вычислений в полях алгебраических чисел, вычислений с решётками, нахождения диофантовых приближений и ряд других вопросов.

Вычислительные машины и электронные средства связи проникли практически во все сферы человеческой деятельности. Немыслима без них и современная криптография. Шифрование и дешифрование текстов можно представлять себе как процессы переработки целых чисел при помощи ЭВМ, а способы, которыми выполняются эти операции, как некоторые функции, определённые на множестве целых чисел. Всё это делает естественным появление в криптографии методов теории чисел. Кроме того, стойкость ряда современных криптосистем обосновывается только сложностью некоторых теоретико-числовых задач (см. [24]).

Но возможности ЭВМ имеют определённые границы. Приходится разбивать длинную цифровую последовательность на блоки ограниченной длины и шифровать каждый такой блок отдельно. Мы будем считать в дальнейшем, что все шифруемые целые числа неотрицательны и по величине меньше некоторого заданного (скажем, техническими ограничениями) числа m. Таким же условиям будут удовлетворять и числа, получаемые в процессе шифрования. Это позволяет считать и те, и другие числа

элементами кольца вычетов $\mathbb{Z}/m\mathbb{Z}$. Шифрующая функция при этом может рассматриваться как взаимнооднозначное отображение колец вычетов

$$f : \mathbb{Z}/m\mathbb{Z} \to \mathbb{Z}/m\mathbb{Z},$$

а число $f(x)$ представляет собой сообщение x в зашифрованном виде.

Простейший шифр такого рода — шифр замены (см. [1]), соответствует отображению $f : x \to x + k \pmod m$ при некотором фиксированном целом k. Подобный шифр использовал ещё Юлий Цезарь. Конечно, не каждое отображение f подходит для целей надежного сокрытия информации, см. [1].

В 1978 г., см. [2], американцы Р. Ривест, А. Шамир и Л. Адлеман (R.L.Rivest, A.Shamir, L.Adleman) предложили пример функции f, обладающей рядом замечательных достоинств. На её основе была построена реально используемая система шифрования, получившая название по первым буквам имен авторов — система RSA. Эта функция такова, что

а) существует достаточно быстрый алгоритм вычисления значений $f(x)$;

б) существует достаточно быстрый алгоритм вычисления значений обратной функции $f^{-1}(x)$;

в) функция $f(x)$ обладает некоторым «секретом», знание которого позволяет быстро вычислять значения $f^{-1}(x)$; в противном же случае вычисление $f^{-1}(x)$ становится трудно разрешимой в вычислительном отношении задачей, требующей для своего решения столь много времени, что по его прошествии зашифрованная информация перестает представлять интерес для лиц, использующих отображение f в качестве шифра.

Подробнее об отображениях такого сорта и возможностях их использования в криптографии рассказано в [1,9].

Еще до выхода из печати статьи [2] копия доклада в Массачусетском Технологическом институте, посвящённого системе RSA, была послана известному популяризатору математики М. Гарднеру, который в 1977 г. в журнале Scientific American опубликовал статью [3], посвящённую этой системе шифрования. В русском переводе заглавие статьи Гарднера звучит так: *Новый вид шифра, на расшифровку которого потребуются миллионы лет*. Именно статья [3] сыграла важнейшую роль в распространении информации об RSA, привлекла к криптографии внимание широких кругов неспециалистов и фактически способствовала бурному прогрессу этой области, произошедшему в последовавшие 20 лет.

1. Система шифрования RSA

В дальнейшем мы будем предполагать, что читатель знаком с элементарными фактами теории чисел. Тех же, кто хотел бы ознакомиться с ними или напомнить себе эти факты, мы отсылаем к книге [4].

Пусть m и ϵ натуральные числа. Функция f, реализующая схему RSA, устроена следующим образом

$$f : x \to x^e \pmod{m}. \tag{1}$$

Для расшифровки сообщения $a = f(x)$ достаточно решить сравнение

$$x^e \equiv a \pmod{m}. \tag{2}$$

При некоторых условиях на m и ϵ это сравнение имеет единственное решение x.

Для того, чтобы описать эти условия и объяснить, как можно найти решение, нам потребуется одна теоретико-числовая функция, так называемая функция Эйлера. Эта функция натурального аргумента m обозначается $\varphi(m)$ и равняется количеству целых чисел на отрезке от 1 до m, взаимно простых с m. Так $\varphi(1) = 1$ и $\varphi(p^r) = p^{r-1}(p-1)$ для любого простого числа p и натурального r. Кроме того, $\varphi(ab) = \varphi(a)\varphi(b)$ для любых натуральных взаимно простых a и b. Эти свойства позволяют легко вычислить значение $\varphi(m)$, если известно разложение числа m на простые сомножители.

Если показатель степени ϵ в сравнении (2) взаимно прост с $\varphi(m)$, то сравнение (2) имеет единственное решение. Для того, чтобы найти его, определим целое число d, удовлетворяющее условиям

$$d\epsilon \equiv 1 \pmod{\varphi(m)}, \quad 1 \leqslant d < \varphi(m). \tag{3}$$

Такое число существует, поскольку $(\epsilon, \varphi(m)) = 1$, и притом единственно. Здесь и далее символом (a, b) будет обозначаться наибольший общий делитель чисел a и b. Классическая теорема Эйлера, см. [4], утверждает, что для каждого числа x, взаимно простого с m, выполняется сравнение $x^{\varphi(m)} \equiv 1 \pmod{m}$ и, следовательно,

$$a^d \equiv x^{d\epsilon} \equiv x \pmod{m}. \tag{4}$$

Таким образом, в предположении $(a, m) = 1$, единственное решение сравнения (2) может быть найдено в виде

$$x \equiv a^d \pmod{m}. \tag{5}$$

Если дополнительно предположить, что число m состоит из различных простых сомножителей, то сравнение (5) будет выполняться и без предположения $(a, m) = 1$. Действительно, обозначим $r = (a, m)$ и $s = m/r$. Тогда

$\varphi(m)$ делится на $\varphi(s)$, а из (2) следует, что $(x, s) = 1$. Подобно (4), теперь легко находим $x \equiv a^d \pmod{s}$. А кроме того, имеем $x \equiv 0 \equiv a^d \pmod{r}$. Получившиеся сравнения в силу $(r, s) = 1$ дают нам (5).

Функция (1), принятая в системе RSA, может быть вычислена достаточно быстро. Как это сделать, мы обсудим чуть ниже. Пока отметим лишь, что обратная к $f(x)$ функция $f^{-1} : x \to x^d \pmod{m}$ вычисляется по тем же правилам, что и $f(x)$, лишь с заменой показателя степени ϵ на d. Таким образом, для функции (1) будут выполнены указанные выше свойства а) и б).

Для вычисления функции (1) достаточно знать лишь числа ϵ и m. Именно они составляют открытый ключ для шифрования. А вот для вычисления обратной функции требуется знать число d, оно и является «секретом», о котором речь идёт в пункте в). Казалось бы, ничего не стоит, зная число m, разложить его на простые сомножители, вычислить затем с помощью известных правил значение $\varphi(m)$ и, наконец, с помощью (3) определить нужное число d. Все шаги этого вычисления могут быть реализованы достаточно быстро, за исключением первого. Именно разложение числа m на простые множители и составляет наиболее трудоемкую часть вычислений. В теории чисел несмотря на многолетнюю её историю и на очень интенсивные поиски в течение последних 20 лет, эффективный алгоритм разложения натуральных чисел на множители так и не найден. Конечно, можно, перебирая все простые числа до $\sqrt{m}$, и, деля на них m, найти требуемое разложение. Но, учитывая, что количество простых в этом промежутке, асимптотически равно $2\sqrt{m} \cdot (\ln m)^{-1}$, см. [5], гл. 5, находим, что при m, записываемом 100 десятичными цифрами, найдётся не менее $4 \cdot 10^{42}$ простых чисел, на которые придётся делить m при разложении его на множители. Очень грубые прикидки показывают, что компьютеру, выполняющему миллион делений в секунду, для разложения числа $m > 10^{99}$ таким способом на простые сомножители потребуется не менее, чем 10^{35} лет. Известны и более эффективные способы разложения целых чисел на множители, чем простой перебор простых делителей, но и они работают очень медленно. Таким образом, название статьи М. Гарднера вполне оправдано.

Авторы схемы RSA предложили выбирать число m в виде произведения двух простых множителей p и q, примерно одинаковых по величине. Так как

$$\varphi(m) = \varphi(pq) = (p - 1)(q - 1), \tag{6}$$

то единственное условие на выбор показателя степени ϵ в отображении

(1) есть

$$(\epsilon, p-1) = (\epsilon, q-1) = 1. \tag{7}$$

Итак, лицо, заинтересованное в организации шифрованной переписки с помощью схемы RSA, выбирает два достаточно больших простых числа p и q. Перемножая их, оно находит число $m = pq$. Затем выбирается число ϵ, удовлетворяющее условиям (7), вычисляется с помощью (6) число $\varphi(m)$ и с помощью (3) — число d. Числа m и ϵ публикуются, число d остается секретным. Теперь любой может отправлять зашифрованные с помощью (1) сообщения организатору этой системы, а организатор легко сможет расшифровывать их с помощью (5).

Для иллюстрации своего метода Ривест, Шамир и Адлеман зашифровали таким способом некоторую английскую фразу. Сначала она стандартным образом (a=01, b=02, ..., z=26, пробел=00) была записана в виде целого числа x, а затем зашифрована с помощью отображения (1) при

$$m = 11438162575788886766932577997614661201021829672124236256256184293570$$
$$693524573389783059712356395870505898907514759929002687954354 1$$

и $\epsilon = 9007$. Эти два числа были опубликованы, причем дополнительно сообщалось, что $m = pq$, где p и q — простые числа, записываемые соответственно 64 и 65 десятичными знаками. Первому, кто расшифрует соответствующее сообщение

$$f(x) = 968696137546220614771409222543558829057599911245743198746951209308$$
$$162982251457083569314766228839896280133919905518299451578151 54,$$

была обещана награда в 100\$.

Эта история завершилась спустя 17 лет в 1994 г., см. [6], когда D. Atkins, M. Graff, A. K. Lenstra и P. C. Leyland сообщили о расшифровке фразы, предложенной в [2]. Она[1)] была вынесена в заголовок статьи [6], а соответствующие числа p и q оказались равными

$$p = 3490529510847650949147849619903898133417764638493387843990820577,$$
$$q = 3276913299326670954996198819083446141317764296799294253979828853 3.$$

Интересующиеся могут найти детали вычислений в работе [6]. Здесь же мы отметим, что этот замечательный результат (разложение на множители 129-значного десятичного числа) был достигнут благодаря использованию алгоритма разложения чисел на множители, называемого

[1)] ***The magic words are squeamish ossifrage.*** Приведём перевод двух последних слов, входящих в эту, по всей видимости, бессмысленную фразу:
squeamish — брезгливый, привередливый, обидчивый;
ossifrage — скопа (вид птицы типа выпи).

методом квадратичного решета. Выполнение вычислений потребовало колоссальных ресурсов. В работе, возглавлявшейся четырьмя авторами проекта, и продолжавшейся после предварительной теоретической подготовки примерно 220 дней, на добровольных началах участвовало около 600 человек и примерно 1600 компьютеров, объединённых сетью Internet. Наконец, отметим, что премия в 100$ была передана в Free Software Foundation.

Описанная выше схема RSA ставит ряд вопросов, которые мы и попробуем обсудить ниже. Например, как проводить вычисления с большими числами, ведь стандартное математическое обеспечение не позволяет перемножать числа размером по 65 десятичных знаков? Как вычислять огромные степени больших чисел? Что значит быстрый алгоритм вычисления и что такое сложная вычислительная задача? Где взять большие простые числа? Как, например, построить простое число в 65 десятичных знаков? Существуют ли другие способы решения сравнения (2)? Ведь, если можно найти решение (2), не вычисляя секретный показатель d или не разлагая число m на простые сомножители, да ещё сделать это достаточно быстро, вся система RSA разваливается. Наверное, читателю могут прийти в голову и другие вопросы.

Начнем с конца. За 17 лет, прошедших между публикациями работ [2] и [6], никто так и не смог расшифровать предложенную авторами RSA фразу. Конечно, это всего лишь косвенное подтверждение стойкости системы RSA, но все же достаточно убедительное. Ниже мы обсудим теоретические проблемы, возникающие при решении полиномиальных сравнений.

Мы не будем обсуждать, как выполнять арифметические действия с большими целыми числами, рекомендуем читателю обратиться к замечательной книжке Д. Кнута [7, гл. 4]. Заметим только, что большое число всегда можно разбить на меньшие блоки, с которыми компьютер может оперировать так же, как мы оперируем с цифрами, когда проводим вычисления вручную на бумаге. Конечно, для этого нужны специальные программы. Созданы и получили достаточно широкое распространение даже специальные языки программирования для вычислений с большими числами. Укажем здесь два из них — PARI и UBASIC. Эти языки свободно распространяются. Информацию о том, как их получить в пользование, можно найти в книге [19].

2. Сложность теоретико-числовых алгоритмов

Сложность алгоритмов теории чисел обычно принято измерять количеством арифметических операций (сложений, вычитаний, умножений и делений с остатком), необходимых для выполнения всех действий, предписанных алгоритмом. Впрочем, это определение не учитывает величины чисел, участвующих в вычислениях. Ясно, что перемножить два стозначных числа значительно сложнее, чем два однозначных, хотя при этом и в том, и в другом случае выполняется лишь одна арифметическая операция. Поэтому иногда учитывают ещё и величину чисел, сводя дело к так называемым битовым операциям, т. е. оценивая количество необходимых операций с цифрами 0 и 1, в двоичной записи чисел. Это зависит от рассматриваемой задачи, от целей автора и т.д.

На первый взгляд странным также кажется, что операции умножения и деления приравниваются по сложности к операциям сложения и вычитания. Житейский опыт подсказывает, что умножать числа значительно сложнее, чем складывать их. В действительности же, вычисления можно организовать так, что на умножение или деление больших чисел понадобится не намного больше битовых операций, чем на сложение. В книге [8] описывается алгоритм Шёнхаге – Штрассена, основанный на так называемом быстром преобразовании Фурье, и требующий $O(n \ln n \ln\ln n)$ битовых операций для умножения двух n-разрядных двоичных чисел. Таким же количеством битовых операций можно обойтись при выполнении деления с остатком двух двоичных чисел, записываемых не более, чем n цифрами. Для сравнения отметим, что сложение n-разрядных двоичных чисел требует $O(n)$ битовых операций.

Говоря в этой статье о сложности алгоритмов, мы будем иметь в виду количество арифметических операций. При построении эффективных алгоритмов и обсуждении верхних оценок сложности обычно хватает интуитивных понятий той области математики, которой принадлежит алгоритм. Формализация же этих понятий требуется лишь тогда, когда речь идёт об отсутствии алгоритма или доказательстве нижних оценок сложности. Более детальное и формальное обсуждение этих вопросов см. в статье [9].

Приведем теперь примеры достаточно быстрых алгоритмов с оценками их сложности. Здесь и в дальнейшем мы не будем придерживаться формального описания алгоритмов, стараясь в первую очередь объяснить смысл выполняемых действий.

Следующий алгоритм вычисляет $a^d \pmod m$ за $O(\ln m)$ арифметических операций. При этом, конечно, предполагается, что натуральные числа a и d не превосходят по величине m.

1. Алгоритм вычисления $a^d \pmod m$.

1) Представим d в двоичной системе счисления $d = d_0 2^r + \cdots + d_{r-1}2 + d_r$, где d_i, цифры в двоичном представлении, равны 0 или 1, $d_0 = 1$.

2) Положим $a_0 = a$ и затем для $i = 1, \ldots, r$ вычислим

$$a_i \equiv a_{i-1}^2 \cdot a^{d_i} \pmod m.$$

3) a_r есть искомый вычет $a^d \pmod m$.

Справедливость этого алгоритма вытекает из сравнения

$$a_i \equiv a^{d_0 2^i + \cdots + d_i} \pmod m,$$

легко доказываемого индукцией по i.

Так как каждое вычисление на шаге 2 требует не более трёх умножений по модулю m и этот шаг выполняется $r \leqslant \log_2 m$ раз, то сложность алгоритма может быть оценена величиной $O(\ln m)$.

Второй алгоритм — это классический алгоритм Евклида вычисления наибольшего общего делителя целых чисел. Мы предполагаем заданными два натуральных числа a и b и вычисляем их наибольший общий делитель (a, b).

2. Алгоритм Евклида.

1) Вычислим r — остаток от деления числа a на b, $a = bq + r$, $0 \leqslant r < b$.

2) Если $r = 0$, то b есть искомое число.

3) Если $r \neq 0$, то заменим пару чисел $\langle a, b\rangle$ парой $\langle b, r\rangle$ и перейдем к шагу 1.

Не останавливаясь на объяснении, почему алгоритм действительно находит (a, b), это общеизвестно, докажем некоторую оценку его сложности.

Теорема 1. *При вычислении наибольшего общего делителя (a, b) с помощью алгоритма Евклида будет выполнено не более $5p$ операций деления с остатком, где p есть количество цифр в десятичной записи меньшего из чисел a и b.*

Доказательство. Положим $r_0 = a > b$ и определим $r_1, r_2, \ldots, r_n$ — последовательность делителей, появляющихся в процессе выполнения шага 1 алгоритма Евклида. Тогда

$$r_1 = b, \ldots, \quad 0 \leqslant r_{i+1} < r_i, \quad i = 0, 1, \ldots n-1.$$

Пусть также $u_0 = 1$, $u_1 = 1$, $u_{k+1} = u_k + u_{k-1}$, $k \geqslant 1$, — последовательность Фибоначчи. Индукцией по i от $i = n-1$ до $i = 0$ легко доказывается неравенство $r_{i+1} \geqslant u_{n-i}$. А так как $u_n \geqslant 10^{(n-1)/5}$, то имеем неравенства $10^p > b = r_1 \geqslant u_n \geqslant 10^{(n-1)/5}$ и $n < 5p + 1$.

Немного подправив алгоритм Евклида, можно достаточно быстро решать сравнения $ax \equiv 1 \pmod{b}$ при условии, что $(a, b) = 1$. Эта задача равносильна поиску целых решений уравнения $ax + by = 1$.

3. Алгоритм решения уравнения $ax + by = 1$.

0) Определим матрицу $E = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$.

1) Вычислим r *— остаток от деления числа* a *на* b, $a = bq + r$, $0 \leqslant r < b$.

2) Если $r = 0$, *то второй столбец матрицы* E *даёт вектор* $\begin{pmatrix} x \\ y \end{pmatrix}$ *решений уравнения.*

3) Если $r \neq 0$, *то заменим матрицу* E *матрицей* $E \cdot \begin{pmatrix} 0 & 1 \\ 1 & -q \end{pmatrix}$.

4) Заменим пару чисел $\langle a, b \rangle$ *парой* $\langle b, r \rangle$ *и перейдем к шагу 1.*

Если обозначить через E_k матрицу E, возникающую в процессе работы алгоритма перед шагом 2 после k делений с остатком (шаг 1), то в обозначениях из доказательства теоремы 1 в этот момент выполняется векторное равенство $\langle a, b \rangle \cdot E_k = \langle r_{k-1}, r_k \rangle$. Его легко доказать индукцией по k. Поскольку числа a и b взаимно просты, имеем $r_n = 1$, и это доказывает, что алгоритм действительно даёт решение уравнения $ax + by = 1$. Буквой n мы обозначили количество делений с остатком, которое в точности такое же, как и в алгоритме Евклида.

Три приведённых выше алгоритма относятся к разряду так называемых полиномиальных алгоритмов. Это название носят алгоритмы, сложность которых оценивается сверху степенным образом в зависимости от длины записи входящих чисел (см. подробности в [9]). Если наибольшее из чисел, подаваемых на вход алгоритма, не превосходит m, то сложность алгоритмов этого типа оценивается величиной $O(\ln^c m)$, где c — некоторая абсолютная постоянная. Во всех приведённых выше примерах $c = 1$.

Полиномиальные алгоритмы в теории чисел — большая редкость. Да и оценки сложности алгоритмов чаще всего опираются на какие-либо не доказанные, но правдоподобные гипотезы, обычно относящиеся к аналитической теории чисел.

Для некоторых задач эффективные алгоритмы вообще не известны. Иногда в таких случаях все же можно предложить последовательность действий, которая, «если повезет», быстро приводит к требуемому результату. Существует класс так называемых вероятностных алгоритмов, которые дают правильный результат, но имеют вероятностную оценку времени работы. Обычно работа этих алгоритмов зависит от одного или нескольких параметров. В худшем случае они работают достаточно

долго. Но удачный выбор параметра определяет быстрое завершение работы. Такие алгоритмы, если множество «хороших» значений параметров велико, на практике работают достаточно эффективно, хотя и не имеют хороших оценок сложности.

Мы будем иногда использовать слова детерминированный алгоритм, чтобы отличать алгоритмы в обычном смысле от вероятностных алгоритмов.

Как пример, рассмотрим вероятностный алгоритм, позволяющий эффективно находить решения полиномиальных сравнений по простому модулю. Пусть p — простое число, которое предполагается большим, и $f(x) \in \mathbb{Z}[x]$ — многочлен, степень которого предполагается ограниченной. Задача состоит в отыскании решений сравнения

$$f(x) \equiv 0 \pmod{p}. \tag{8}$$

Например, речь может идти о решении квадратичных сравнений, если степень многочлена $f(x)$ равна 2. Другими словами, мы должны отыскать в поле $\mathbb{F}_p = \mathbb{Z}/p\mathbb{Z}$ все элементы, удовлетворяющие уравнению $f(x) = 0$.

Согласно малой теореме Ферма, все элементы поля $\mathbb{F}_p$ являются однократными корнями многочлена $x^p - x$. Поэтому, вычислив наибольший общий делитель $d(x) = (x^p - x, f(x))$, мы найдем многочлен $d(x)$, множество корней которого в поле $\mathbb{F}_p$ совпадает с множеством корней многочлена $f(x)$, причем все эти корни однократны. Если окажется, что многочлен $d(x)$ имеет нулевую степень, т. е. лежит в поле $\mathbb{F}_p$, это будет означать, что сравнение (8) не имеет решений.

Для вычисления многочлена $d(x)$ удобно сначала вычислить многочлен $c(x) \equiv x^p \pmod{f(x)}$, пользуясь алгоритмом, подобным описанному выше алгоритму возведения в степень (напомним, что число p предполагается большим). А затем с помощью аналога алгоритма Евклида вычислить $d(x) = (c(x) - x, f(x))$. Всё это выполняется за полиномиальное количество арифметических операций.

Таким образом, обсуждая далее задачу нахождения решений сравнения (8), мы можем предполагать, что в кольце многочленов $\mathbb{F}_p[x]$ справедливо равенство

$$f(x) = (x - a_1) \cdot \ldots \cdot (x - a_n), \quad a_i \in \mathbb{F}_p, a_i \neq a_j.$$

4. Алгоритм нахождения делителей многочлена $f(x)$ в кольце $\mathbb{F}_p[x]$.

1) Выберем каким-либо способом элемент $\delta \in \mathbb{F}_p$.

2) Вычислим наибольший общий делитель $g(x) = (f(x), (x+\delta)^{\frac{p-1}{2}} - 1)$.

3) Если многочлен $g(x)$ окажется собственным делителем $f(x)$, то

многочлен $f(x)$ распадётся на два множителя и с каждым из них независимо нужно будет проделать все операции, предписываемые настоящим алгоритмом для многочлена $f(x)$.

4) Если окажется, что $g(x) = 1$ или $g(x) = f(x)$, следует перейти к шагу 1 и, выбрав новое значение δ, продолжить выполнение алгоритма.

Количество операций на шаге 2 оценивается величиной $O(\ln p)$, если вычисления проводить так, как это указывалось выше при нахождении $d(x)$. Выясним теперь, сколь долго придётся выбирать числа δ, пока на шаге 2 не будет найден собственный делитель $f(x)$.

Количество решений уравнения $(t + a_1)^{\frac{p-1}{2}} = (t + a_2)^{\frac{p-1}{2}}$ в поле $\mathbb{F}_p$ не превосходит $\frac{p-3}{2}$. Это означает, что подмножество $D \subset \mathbb{F}_p$, состоящее из элементов δ, удовлетворяющих условиям

$$(\delta + a_1)^{\frac{p-1}{2}} \neq (\delta + a_2)^{\frac{p-1}{2}}, \quad \delta \neq -a_1,\ \delta \neq -a_2,$$

состоит не менее, чем из $\frac{p-1}{2}$ элементов. Учитывая теперь, что каждый ненулевой элемент $b \in \mathbb{F}_p$ удовлетворяет одному из равенств $b^{\frac{p-1}{2}} = 1$, либо $b^{\frac{p-1}{2}} = -1$, заключаем, что для $\delta \in D$ одно из чисел a_1, a_2 будет корнем многочлена $(x+\delta)^{\frac{p-1}{2}} - 1$, а другое — нет. Для таких элементов δ многочлен $g(x)$, определённый на шаге 2 алгоритма, будет собственным делителем многочлена $f(x)$.

Итак, существует не менее $\frac{p-1}{2}$ «удачных» выборов элемента δ, при которых на шаге 2 алгоритма многочлен $f(x)$ распадётся на два собственных множителя. Следовательно, при «случайном» выборе элемента $\delta \in \mathbb{F}_p$, вероятность того, что многочлен не разложится на множители после k повторений шагов алгоритма 1–4, не превосходит 2^{-k}. Вероятность с ростом k убывает очень быстро. И действительно, на практике этот алгоритм работает достаточно эффективно.

Заметим, что при оценке вероятности мы использовали только два корня многочлена $f(x)$. При $n > 2$ эта вероятность, конечно, ещё меньше. Более тонкий анализ с использованием оценок А. Вейля для сумм характеров показывает, что вероятность для многочлена $f(x)$ не распасться на множители при однократном проходе шагов алгоритма 1–4, не превосходит $2^{-n} + O(p^{-1/2})$. Здесь постоянная в $O(\cdot)$ зависит от n. Детали доказательства см. в [26]. В настоящее время известно элементарное доказательство оценки А. Вейля (см. [11]).

В книге [7] описывается принадлежащий Берлекэмпу детерминированный алгоритм решения сравнения (8), требующий $O(pn^3)$ арифметических операций. Ясно, что он практически бесполезен при больших p, а вот при маленьких p и не очень больших n он работает не очень долго.

Если в сравнении (8) заменить простой модуль p составным модулем m, то задача нахождения решений соответствующего сравнения становится намного более сложной. Известные алгоритмы её решения основаны на сведении сравнения к совокупности сравнений (8) по простым модулям — делителям m, и, следовательно, они требуют разложения числа m на простые сомножители, что, как уже указывалось, является достаточно трудоемкой задачей.

3. Как отличить составное число от простого

Существует довольно эффективный способ убедиться, что заданное число является составным, не разлагая это число на множители. Согласно малой теореме Ферма, если число N простое, то для любого целого a, не делящегося на N, выполняется сравнение

$$a^{N-1} \equiv 1 \pmod{N}. \tag{9}$$

Если же при каком-то a это сравнение нарушается, можно утверждать, что N — составное. Проверка (9) не требует больших вычислений, это следует из алгоритма 1. Вопрос только в том, как найти для составного N целое число a, не удовлетворяющее (9). Можно, например, пытаться найти необходимое число a, испытывая все целые числа подряд, начиная с 2. Или попробовать выбирать эти числа случайным образом на отрезке $1 < a < N$.

К сожалению, такой подход не всегда даёт то, что хотелось бы. Имеются составные числа N, обладающие свойством (9) для любого целого a с условием $(a, N) = 1$. Такие числа называются числами Кармайкла. Рассмотрим, например, число $561 = 3 \cdot 11 \cdot 17$. Так как 560 делится на каждое из чисел 2, 10, 16, то с помощью малой теоремы Ферма легко проверить, что 561 есть число Кармайкла. Можно доказать (Carmichael, 1912), что любое из чисел Кармайкла имеет вид $N = p_1 \cdot \ldots \cdot p_r$, $r \geqslant 3$, где все простые p_i различны, причем $N - 1$ делится на каждую разность $p_i - 1$. Лишь недавно, см. [12], была решена проблема о бесконечности множества таких чисел.

В 1976 г. Миллер предложил заменить проверку (9) проверкой несколько иного условия. Детали последующего изложения можно найти в [10]. Если N — простое число, $N - 1 = 2^s \cdot t$, где t нечётно, то согласно малой теореме Ферма для каждого a с условием $(a, N) = 1$ хотя бы одна из скобок в произведении

$$(a^t - 1)(a^t + 1)(a^{2t} + 1) \cdot \ldots \cdot (a^{2^{s-1}t} + 1) = a^{N-1} - 1$$

делится на N. Обращение этого свойства можно использовать, чтобы отличать составные числа от простых.

Пусть N — нечётное составное число, $N-1=2^s\cdot t$, где t нечётно. Назовем целое число a, $1<a<N$, «хорошим» для N, если нарушается одно из двух условий:
α) N не делится на a;
β) $a^t\equiv 1 \pmod N$ или существует целое k, $0\leqslant k<s$, такое, что

$$a^{2^k t}\equiv -1 \pmod N.$$

Из сказанного ранее следует, что для простого числа N не существует хороших чисел a. Если же N составное число, то, как доказал Рабин, их существует не менее $\frac{3}{4}(N-1)$.

Теперь можно построить вероятностный алгоритм, отличающий составные числа от простых.

5. Алгоритм, доказывающий непростоту числа.

1) Выберем случайным образом число a, $1<a<N$, и проверим для этого числа указанные выше свойства α) и β).

2) Если хотя бы одно из них нарушается, то число N составное.

3) Если выполнены оба условия α) и β), возвращаемся к шагу 1.

Из сказанного выше следует, что составное число не будет определено как составное после однократного выполнения шагов 1–3 с вероятностью не большей 4^{-1}. А вероятность не определить его после k повторений не превосходит 4^{-k}, т. е. убывает очень быстро.

Миллер предложил детерминированный алгоритм определения составных чисел, имеющий сложность $O(\ln^3 N)$, однако справедливость его результата зависит от недоказанной в настоящее время так называемой расширенной гипотезы Римана. Согласно этому алгоритму достаточно проверить условия α) и β) для всех целых чисел a, $2\leqslant a\leqslant 70\ln^2 N$. Если при каком-нибудь a из указанного промежутка нарушается одно из условий α) или β), число N составное. В противном случае оно будет простым или степенью простого числа. Последняя возможность, конечно, легко проверяется.

Напомним некоторые понятия, см. [5], необходимые для формулировки расширенной гипотезы Римана. Они понадобятся нам и в дальнейшем. Пусть $m\geqslant 2$ — целое число. Функция $\chi:\mathbb{Z}\to\mathbb{C}$ называется характером Дирихле по модулю m, или просто характером, если эта функция периодична с периодом m, отлична от нуля только на числах, взаимно простых с m, и мультипликативна, т. е. для любых целых u,v выполняется равенство $\chi(uv)=\chi(u)\chi(v)$. Для каждого m существует ровно $\varphi(m)$

характеров Дирихле. Они образуют группу по умножению. Единичным элементом этой группы является так называемый главный характер χ_0, равный 1 на всех числах, взаимно простых с m, и 0 на остальных целых числах. Порядком характера называется его порядок как элемента мультипликативной группы характеров.

С каждым характером может быть связана так называемая L-функция Дирихле — функция комплексного переменного s, определённая рядом $L(s,\chi)=\sum\limits_{n=1}^{\infty}\dfrac{\chi(n)}{n^s}$. Сумма этого ряда аналитична в области $\operatorname{Re} s > 1$ и может быть аналитически продолжена на всю комплексную плоскость. Следующее соотношение $L(s,\chi_0)=\zeta(s)\prod\limits_{p|m}(1-p^{-s})$ связывает L-функцию, отвечающую главному характеру, с дзета-функцией Римана $\zeta(s)=\sum\limits_{n=0}^{\infty}\dfrac{1}{n^s}$. Расширенная гипотеза Римана утверждает, что комплексные нули всех L-функций Дирихле, расположенные в полосе $0<\operatorname{Re} s<1$, лежат на прямой $\operatorname{Re} s=\frac{1}{2}$. В настоящее время не доказана даже простейшая форма этой гипотезы — классическая гипотеза Римана, утверждающая такой же факт о нулях дзета-функции.

В 1952 г. Анкени с помощью расширенной гипотезы Римана доказал, что для каждого простого числа q существует квадратичный невычет a, удовлетворяющий неравенствам $2\leqslant a\leqslant 70\ln^2 q$. Константа 70 была посчитана позднее. Именно это утверждение и лежит в основе алгоритма Миллера. В 1957 г. Берджесс доказал существование такого невычета без использования расширенной гипотезы Римана, но с худшей оценкой $2\leqslant a\leqslant q^{\frac{1}{4\sqrt{e}}}+\varepsilon$, справедливой при любом положительном ε и q, большем некоторой границы, зависящей от ε.

Алгоритм Миллера принципиально отличается от алгоритма 5, так как полученное с его помощью утверждение о том, что число N — составное, опирается на недоказанную расширенную гипотезу Римана и потому может быть неверным. В то время как вероятностный алгоритм 5 даёт совершенно правильный ответ для составных чисел. Несмотря на отсутствие оценок сложности, на практике он работает вполне удовлетворительно.

4. Как строить большие простые числа

Мы не будем описывать здесь историю этой задачи, рекомендуем обратиться к книге [7] и обзорам [10, 11]. Конечно же, большие простые числа можно строить сравнительно быстро. При этом можно обеспечить

их случайное распределение в заданном диапазоне величин. В противном случае теряла бы всякий практический смысл система шифрования RSA. Наиболее эффективным средством построения простых чисел является несколько модифицированная малая теорема Ферма.

Теорема 2. *Пусть N, S — нечётные натуральные числа, $N-1 = S \cdot R$, причем для каждого простого делителя q числа S существует целое число a такое, что*

$$a^{N-1} \equiv 1 \pmod{N}, \quad (a^{\frac{N-1}{q}} - 1, N) = 1. \tag{10}$$

Тогда каждый простой делитель p числа N удовлетворяет сравнению

$$p \equiv 1 \pmod{2S}.$$

Доказательство. Пусть p — простой делитель числа N, а q — некоторый делитель S. Из условий (10) следует, что в поле вычетов $\mathbb{F}_p$ справедливы соотношения

$$a^{N-1} = 1, \; a^{\frac{N-1}{q}} \neq 1, \; a^{p-1} = 1. \tag{11}$$

Обозначим буквой r порядок элемента a в мультипликативной группе поля $\mathbb{F}_p$. Первые два из соотношений (11) означают, что q входит в разложение на простые множители числа r в степени такой же, как и в разложение $N-1$, а последнее — что $p-1$ делится на r. Таким образом, каждый простой делитель числа S входит в разложение $p-1$ в степени не меньшей, чем в S, так что $p-1$ делится на S. Кроме того, $p-1$ чётно. Теорема 2 доказана.

Следствие. *Если выполнены условия теоремы 2 и $R \leqslant 4S+2$, то N — простое число.*

Действительно, пусть N равняется произведению не менее двух простых чисел. Каждое из них, согласно утверждению теоремы 2, не меньше, чем $2S+1$. Но тогда $(2S+1)^2 \leqslant N = SR+1 \leqslant 4S^2+2S+1$. Противоречие и доказывает следствие.

Покажем теперь, как с помощью последнего утверждения, имея большое простое число S, можно построить существенно большее простое число N. Выберем для этого случайным образом чётное число R на промежутке $S \leqslant R \leqslant 4S+2$ и положим $N = SR+1$. Затем проверим число N на отсутствие малых простых делителей, разделив его на малые простые числа; испытаем N некоторое количество раз с помощью алгоритма 5. Если при этом выяснится, что N — составное число, следует выбрать новое значение R и опять повторить вычисления. Так следует делать до

тех пор, пока не будет найдено число N, выдержавшее испытания алгоритмом 5 достаточно много раз. В этом случае появляется надежда на то, что N — простое число, и следует попытаться доказать простоту с помошью тестов теоремы 2.

Для этого можно случайным образом выбирать число a, $1 < a < N$, и проверять для него выполнимость соотношений

$$a^{N-1} \equiv 1 \pmod{N}, \quad (a^R - 1, N) = 1. \tag{12}$$

Если при выбранном a эти соотношения выполняются, то, согласно следствию из теоремы 2, можно утверждать, что число N простое. Если же эти условия нарушаются, нужно выбрать другое значение a и повторять эти операции до тех пор, пока такое число не будет обнаружено.

Предположим, что построенное число N действительно является простым. Зададимся вопросом, сколь долго придётся перебирать числа a, пока не будет найдено такое, для которого будут выполнены условия (12). Заметим, что для простого числа N первое условие (12), согласно малой теореме Ферма, будет выполняться всегда. Те же числа a, для которых нарушается второе условие (12), удовлетворяют сравнению $a^R \equiv 1 \pmod{N}$. Как известно, уравнение $x^R = 1$ в поле вычетов $\mathbb{F}_N$ имеет не более R решений. Одно из них $x = 1$. Поэтому на промежутке $1 < a < N$ имеется не более $R - 1$ чисел, для которых не выполняются условия (12). Это означает, что, выбирая случайным образом числа a на промежутке $1 < a < N$, при простом N можно с вероятностью большей, чем $1 - O(S^{-1})$, найти число a, для которого будут выполнены условия теоремы 2, и тем доказать, что N действительно является простым числом.

Заметим, что построенное таким способом простое число N будет удовлетворять неравенству $N > S^2$, т. е. будет записываться вдвое большим количеством цифр, чем исходное простое число S. Заменив теперь число S на найденное простое число N и повторив с этим новым S все указанные выше действия, можно построить ещё большее простое число. Начав с какого-нибудь простого числа, скажем, записанного 10 десятичными цифрами (простоту его можно проверить, например, делением на маленькие табличные простые числа), и повторив указанную процедуру достаточное число раз, можно построить простые числа нужной величины.

Обсудим теперь некоторые теоретические вопросы, возникающие в связи с нахождением числа R, удовлетворяюшего неравенствам $S \leqslant R \leqslant 4S + 2$, и такого, что $N = SR + 1$ — простое число. Прежде всего, согласно теореме Дирихле, доказанной ещё в 1839 г., прогрессия $2Sn + 1$, $n = 1, 2, 3, \ldots$ содержит бесконечное количество простых чисел. Нас инте-

ресуют простые числа, лежащие недалеко от начала прогрессии. Оценка наименьшего простого числа в арифметической прогрессии была получена в 1944 г. Ю. В. Линником. Соответствующая теорема утверждает, что наименьшее простое число в арифметической прогрессии $2Sn+1$ не превосходит S^C, где C — некоторая достаточно большая абсолютная постоянная. В предположении справедливости расширенной гипотезы Римана можно доказать, [13, стр. 272], что наименьшее такое простое число не превосходит $c(\varepsilon)\cdot S^{2+\varepsilon}$ при любом положительном ε.

Таким образом, в настоящее время никаких теоретических гарантий для существования простого числа $N=SR+1$, $S\leqslant R\leqslant 4S+2$ не существует. Тем не менее опыт вычислений на ЭВМ показывает, что простые числа в арифметической прогрессии встречаются достаточно близко к её началу. Упомянем в этой связи гипотезу о существовании бесконечного количества простых чисел q с условием, что число $2q+1$ также простое, т. е. простым является уже первый член прогрессии.

Очень важен в связи с описываемым методом построения простых чисел также вопрос о расстоянии между соседними простыми числами в арифметической прогрессии. Ведь убедившись, что при некотором R число $N=SR+1$ составное, можно следующее значение R взять равным $R+2$ и действовать так далее, пока не будет найдено простое число N. И если расстояние между соседними простыми числами в прогрессии велико, нет надежды быстро построить нужное число N. Перебор чисел R до того момента, как мы наткнемся на простое число N окажется слишком долгим. В более простом вопросе о расстоянии между соседними простыми числами p_n и p_{n+1} в натуральном ряде доказано лишь, что $p_{n+1}-p_n=O\left(p_n^{\frac{38}{61}+\varepsilon}\right)$, что, конечно, не очень хорошо для наших целей. Вместе с тем существует так называемая гипотеза Крамера (1936 г.), что $p_{n+1}-p_n=O(\ln^2 p_n)$, дающая вполне приемлемую оценку. Примерно такой же результат следует и из расширенной гипотезы Римана. Вычисления на ЭВМ показывают, что простые числа в арифметических прогрессиях расположены достаточно плотно.

В качестве итога обсуждения в этом пункте подчеркнём следующее: если принять на веру, что наименьшее простое число, а также расстояние между соседними простыми числами в прогрессии $2Sn+1$ при $S\leqslant n\leqslant 4S+2$ оцениваются величиной $O(\ln^2 S)$, то описанная схема построения больших простых чисел имеет полиномиальную оценку сложности. Кроме того, несмотря на отсутствие теоретических оценок времени работы алгоритмов, отыскивающих простые числа в арифметических прогрессиях со сравнительно большой разностью, на практике эти алгоритмы

работают вполне удовлетворительно. На обычном персональном компьютере без особых затрат времени строятся таким способом простые числа порядка 10^{300}.

Конечно, способ конструирования простых чисел для использования в схеме RSA должен быть массовым, а сами простые числа должны быть в каком-то смысле хорошо распределёнными. Это вносит ряд дополнительных осложнений в работу алгоритмов. Впрочем, описанная схема допускает массу вариаций. Все эти вопросы рассматриваются в статье [14].

Наконец, отметим, что существуют методы построения больших простых чисел, использующие не только простые делители $N-1$, но и делители чисел $N+1$, N^2+1, $N^2 \pm N+1$. В основе их лежит использование последовательностей целых чисел, удовлетворяющих линейным рекуррентным уравнениям различных порядков. Отметим, что последовательность a^n, члены которой присутствуют в формулировке малой теоремы Ферма, составляет решение рекуррентного уравнения первого порядка $u_{n+1} = au_n$, $u_0 = 1$.

5. Как проверить большое число на простоту

Есть некоторое отличие в постановках задач предыдущего и настоящего пунктов. Когда мы строим простое число N, мы обладаем некоторой дополнительной информацией о нем, возникающей в процессе построения. Например, такой информацией является знание простых делителей числа $N-1$. Эта информация иногда облегчает доказательство простоты N.

В этом пункте мы предполагаем лишь, что нам задано некоторое число N, например, выбранное случайным образом на каком-то промежутке, и требуется установить его простоту, или доказать, что оно является составным. Эту задачу за полиномиальное количество операций решает указанный в п. 3 алгоритм Миллера. Однако, справедливость полученного с его помощью утверждения зависит от недоказанной расширенной гипотезы Римана. Если число N выдержало испытания алгоритмом 5 для 100 различных значений параметра a, то, по-видимому, можно утверждать, что оно является простым с вероятностью большей, чем $1 - 4^{-100}$. Эта вероятность очень близка к единице, однако всё же оставляет некоторую тень сомнения на простоте числа N. В дальнейшем в этом пункте мы будем считать, что заданное число N является простым, а нам требуется лишь доказать это.

В настоящее время известны детерминированные алгоритмы различной сложности для доказательства простоты чисел. Мы остановимся подробнее на одном из них, предложенном в 1983 г. в совместной работе

Адлемана, Померанца и Рамели [15]. Для доказательства простоты или непростоты числа N этот алгоритм требует $(\ln N)^{c\ln\ln\ln N}$ арифметических операций. Здесь c — некоторая положительная абсолютная постоянная. Функция $\ln\ln\ln N$ хоть и медленно, но всё же возрастает с ростом N, поэтому алгоритм не является полиномиальным. Но всё же его практические реализации позволяют достаточно быстро тестировать числа на простоту. Существенные усовершенствования и упрощения в первоначальный вариант алгоритма были внесены в работах Х. Ленстры и А. Коена [16, 17]. Мы будем называть описываемый ниже алгоритм алгоритмом Адлемана – Ленстры.

В основе алгоритма лежит использование сравнений типа малой теоремы Ферма, но в кольцах целых чисел круговых полей, т. е. полей, порождённых над полем $\mathbb{Q}$ числами $\zeta_p = e^{2\pi i/p}$ — корнями из 1. Пусть q — простое нечётное число и c — первообразный корень по модулю q, т. е. образующий элемент мультипликативной группы поля $\mathbb{F}_q$, которая циклична. Для каждого целого числа x, не делящегося на q, можно определить его индекс, $\operatorname{ind}_q x \in \mathbb{Z}/(q-1)\mathbb{Z}$, называемый также *дискретным логарифмом*, с помощью сравнения $x \equiv c^{\operatorname{ind}_q x} \pmod q$. Рассмотрим далее два простых числа p, q с условием, что $q-1$ делится на p, но не делится на p^2.

Следующая функция, определённая на множестве целых чисел,

$$\chi(x) = \begin{cases} 0, \text{ если } & q|x, \\ \zeta_p^{\operatorname{ind}_q x}, \text{ если } & (x,q) = 1 \end{cases}$$

является характером по модулю q и порядок этого характера равен p. Сумма

$$\tau(\chi) = -\sum_{x=1}^{q-1} \chi(x)\zeta_q^x \in \mathbb{Z}[\zeta_p, \zeta_q]$$

называется суммой Гаусса. Формулируемая ниже теорема 3 представляет собой аналог малой теоремы Ферма, используемый в алгоритме Адлемана – Ленстры.

Теорема 3. *Пусть N — нечётное простое число, $(N, pq) = 1$. Тогда в кольце $\mathbb{Z}[\zeta_p, \zeta_q]$ выполняется сравнение*

$$\tau(\chi)^N \equiv \chi(N)^{-N} \cdot \tau(\chi^N) \pmod{NZ[\zeta_p, \zeta_q]}.$$

Если при каких-либо числах p, q сравнение из теоремы 3 нарушается, можно утверждать, что N составное число. В противном случае, если сравнение выполняется, оно даёт некоторую информацию о возможных простых делителях числа N. Собрав такую информацию для различных

p, q, в конце концов удаётся установить, что N имеет лишь один простой делитель и является простым.

В случае $p = 2$ легко проверить, что сравнение из теоремы 3 равносильно хорошо известному в элементарной теории чисел сравнению

$$q^{\frac{N-1}{2}} \equiv \left(\frac{q}{N}\right) \pmod{N}, \tag{13}$$

где $\left(\frac{q}{N}\right)$ — так называемый символ Якоби. Хорошо известно также, что последнее сравнение выполняется не только для простых q, но и для любых целых q, взаимно простых с N. Заметим также, что для вычисления символа Якоби существует быстрый алгоритм, основанный на законе взаимности Гаусса и, в некотором смысле, подобный алгоритму Евклида вычисления наибольшего общего делителя. Следующий пример показывает, каким образом выполнимость нескольких сравнений типа (13) даёт некоторую информацию о возможных простых делителях числа N.

Пример (Х. Ленстра). Пусть N — натуральное число, $(N, 6) = 1$, для которого выполнены сравнения

$$a^{\frac{N-1}{2}} \equiv \left(\frac{a}{N}\right) \pmod{N}, \quad \text{при } a = -1, 2, 3, \tag{14}$$

а кроме того с некоторым целым числом b имеем

$$b^{\frac{N-1}{2}} \equiv -1 \pmod{N}. \tag{15}$$

Как уже указывалось, при простом N сравнения (14) выполняются для любого a, взаимно простого с N, а сравнение (15) означает, что b есть первообразный корень по модулю N. Количество первообразных корней равно $\varphi(N-1)$, т. е. достаточно велико. Таким образом, число b с условием (15) при простом N может быть найдено достаточно быстро с помощью случайного выбора и последующей проверки (15).

Докажем, что из выполнимости (14–15) следует, что каждый делитель r числа N удовлетворяет одному из сравнений

$$r \equiv 1 \pmod{24} \text{ или } r \equiv N \pmod{24}. \tag{16}$$

Не уменьшая общности, можно считать, что r — простое число. Введем теперь обозначения $N - 1 = u \cdot 2^k$, $r - 1 = v \cdot 2^m$, где u и v — нечётные числа. Из (15) и сравнения $b^{r-1} \equiv 1 \pmod{r}$ следует, что $m \geqslant k$. Далее, согласно (14), выполняются следующие сравнения

$$\left(\frac{a}{N}\right) = \left(\frac{a}{N}\right)^v \equiv a^{uv2^{k-1}} \pmod{r}, \qquad \left(\frac{a}{r}\right) = \left(\frac{a}{r}\right)^u \equiv a^{uv2^{m-1}} \pmod{r},$$

означающие (в силу того, что символ Якоби может равняться лишь -1 или $+1$), что

$$\left(\frac{a}{N}\right)^{2^{m-k}} = \left(\frac{a}{r}\right).$$

При $m > k$ это равенство означает, что $\left(\frac{a}{r}\right) = 1$ при $a = -1, 2, 3$, и, следовательно, $r \equiv 1 \pmod{24}$. Если же $m = k$, то имеем $\left(\frac{a}{N}\right) = \left(\frac{a}{r}\right)$ и $r \equiv N \pmod{24}$. Этим (16) доказано.

Информация такого рода получается и в случае произвольных простых чисел p и q с указанными выше свойствами.

Опишем (очень грубо) схему алгоритма Адлемана – Ленстры для проверки простоты N:

1) выбираются различные простые числа $p_1, \ldots, p_k$ и различные простые нечётные $q_1, \ldots, q_s$ такие, что

а) для каждого j все простые делители числа $q_j - 1$ содержатся среди $p_1, \ldots, p_k$ и $q_j - 1$ не делятся на квадрат простого числа;

б) $S = 2q_1 \cdot \ldots \cdot q_s > \sqrt{N}$.

2) для каждой пары выбранных чисел p, q проводятся тесты, подобные сравнению из теоремы 3. Если N не удовлетворяет какому-либо из этих тестов — оно составное. В противном случае

3) определяется не очень большое множество чисел, с которыми только и могут быть сравнимы простые делители N. А именно, каждый простой делитель r числа N должен удовлетворять сравнению вида

$$r \equiv N^j \pmod{S}, \quad 0 \leqslant j < T = p_1 \cdot \ldots \cdot p_k.$$

4) проверяется, содержит ли найденное множество делители N. Если при этом делители не обнаружены, утверждается, что N — простое число.

Если число N составное, оно обязательно имеет простой делитель r, меньший $\sqrt{N} < S$, который сам содержится среди возможных остатков. Именно на этом свойстве основано применение пункта 4) алгоритма.

Пример. Если выбрать следующие множества простых чисел

$$\{p\} = \{2, 3, 5, 7\} \text{ и } \{q\} = \{3, 7, 11, 31, 43, 71, 211\},$$

то таким способом удается проверять простоту чисел $N < 8{,}5 \cdot 10^{19}$.

Отметим, что в работе [15] для тестирования использовались не сравнения теоремы 3, а закон взаимности для степенных вычетов и так

называемые суммы Якоби. Сумма Якоби

$$J(\chi_1,\chi_2)=-\sum_{x=2}^{q-1}\chi_1(x)\chi_2(1-x)$$

определяется для двух характеров χ_1,χ_2 по модулю q. Если характеры имеют порядок p, то соответствующая сумма Якоби принадлежит кольцу $\mathbb{Z}[\zeta_p]$. Поскольку числа p, участвующие в алгоритме, сравнительно невелики, то вычисления с суммами Якоби производятся в полях существенно меньшей степени, чем вычисления с суммами Гаусса. Это главная причина, по которой суммы Якоби предпочтительнее для вычислений. При $\chi_1\chi_2\neq\chi_0$ выполняется классическое соотношение

$$J(\chi_1,\chi_2)=\frac{\tau(\chi_1)\cdot\tau(\chi_2)}{\tau(\chi_1\cdot\chi_2)},$$

связывающее суммы Гаусса с суммами Якоби и позволяющее переписать сравнение теоремы 3 в терминах сумм Якоби (см. [16]). Так, при $p=3$ и $q=7$ соответствующее сравнение, справедливое для простых N, отличных от 2, 3, 7, принимает вид

$$(-3\zeta-2)^{\left[\frac{N}{3}\right]}\cdot(3\zeta+1)^{\left[\frac{2N}{3}\right]}\equiv\xi\ (\mathrm{mod}\ N\mathbb{Z}[\zeta]),$$

где $\zeta=\epsilon^{2\pi i/3}$ и ξ — некоторый корень кубический из 1.

В 1984 г. в работе [17] было внесено существенное усовершенствование в алгоритм, позволившее освободиться от требования неделимости чисел $q-1$ на квадраты простых чисел. В результате, например, выбрав число $T=2^4\cdot3^2\cdot5\cdot7=5040$ и взяв S равным произведению простых чисел q с условием, что T делится на $q-1$, получим $S>1{,}5\cdot10^{52}$, что позволяет доказывать простоту чисел N, записываемых сотней десятичных знаков. При этом вычисления будут проводиться в полях, порождённых корнями из 1 степеней 16, 9, 5 и 7.

Мой персональный компьютер с процессором Pentium-150, пользуясь реализацией этого алгоритма на языке UBASIC, доказал простоту записываемого 65 десятичными знаками, большего из простых чисел в примере Ривеста, Шамира и Адлемана (см. пункт 1) за 8 секунд. Сравнение этих 8 секунд и 17 лет, потребовавшихся для разложения на множители предложенного в примере числа, конечно, впечатляет.

Отметим, что оценка сложности этого алгоритма представляет собой трудную задачу аналитической теории чисел. Как уже указывалось, количество операций оценивается величиной $(\ln N)^{c\ln\ln\ln N}$. Однако соответствующие числа S и T, возникающие в процессе доказательства, не

могут быть явно указаны в зависимости от N. Доказано лишь существование чисел S и T, для которых достигается оценка. Впрочем, есть вероятностный вариант алгоритма, доказывающий простоту простого числа N с вероятностью большей $1-2^{-k}$ за $O(k(\ln N)^{c\ln\ln\ln N})$ арифметических операций. А в предположении расширенной гипотезы Римана эта оценка сложности может быть получена при эффективно указанных S и T.

6. Как раскладывают составные числа на множители

Мы лишь кратко коснемся этой темы, отсылая читателей к книгам [7, 18, 19]. Среди многих алгоритмов разложения мы выберем ту линию развития, которая привела к разложению числа, предложенного RSA.

Поиском эффективных способов разложения целых чисел на множители занимаются уже очень давно. Эта задача интересовала выдающихся учёных в области теории чисел. Вероятно Ферма был первый, кто предложил представить разлагаемое число N в виде разности квадратов $N=x^2-y^2$, а затем, вычисляя $(N, x-y)$, попытаться найти нетривиальный делитель N. Он же предложил и способ, позволяющий найти требуемое представление. Если разлагаемое число имеет два не очень отличающиеся по величине множителя, этот способ позволяет определить их быстрее, чем простой перебор делителей. Лежандр обратил внимание на то, что при таком подходе достаточно получить сравнение

$$x^2 \equiv y^2 \pmod N. \tag{17}$$

Конечно, не каждая пара чисел, удовлетворяющих ему, позволяет разложить N на множители. Эйлер и Гаусс предложили некоторые способы нахождения чисел, связанных соотношением (17). Лежандр использовал для этой цели непрерывные дроби.

Напомним, что каждому иррациональному числу ξ может быть поставлена в соответствие бесконечная последовательность целых чисел $[b_0; b_1, b_2, \ldots]$, называемая его непрерывной дробью. Это сопоставление строится следующим образом

$$x_0=\xi,\ b_i=[x_i],\ x_{i+1}=\frac{1}{x_i-b_i},\ i=0,1,2,\ldots.$$

Лежандр доказал, что непрерывная дробь квадратичной иррациональности периодична. Если раскладывать в непрерывную дробь число $\xi=\sqrt{N}$, то возникающие в процессе разложения числа x_i имеют вид $x_i=\dfrac{\sqrt{N}+P_i}{Q_i}$ с целыми P_i, Q_i, причем всегда $0\leqslant P_i<\sqrt{N}$, $0<Q_i<2\sqrt{N}$. С каждой непрерывной дробью можно связать последовательность рациональных

чисел, так называемых подходящих дробей, $\dfrac{A_i}{B_i}$, $i \geqslant 0$, вычисляемых по правилам

$$A_{i+1} = b_{i+1}A_i + A_{i-1},\ B_{i+1} = b_{i+1}B_i + B_{i-1},\ i \geqslant 0,$$
$$A_0 = b_0,\ B_0 = A_{-1} = 1,\ B_{-1} = 0$$

и стремящихся к разлагаемому числу. Если в непрерывную дробь разлагается число $\xi = \sqrt{N}$, то справедливо соотношение

$$A_{i-1}^2 - NB_{i-1}^2 = (-1)^i Q_i, \tag{18}$$

из которого следует

$$A_{i-1}^2 \equiv (-1)^i Q_i \pmod{N}. \tag{19}$$

Заметим, что длина периода разложения в непрерывную дробь числа $\xi = \sqrt{N}$ может быть большой и достигать величин порядка $\sqrt{N}$.

В 1971 г. Шенкс предложил использовать сравнения (19) для конструирования чисел, удовлетворяющих (17). Если вычисления проводить до тех пор, пока при чётном i не получится $Q_i = R^2$ при некотором целом R, то пара чисел $\langle A_{i-1}, R\rangle$ будет удовлетворять (17) и с её помошью можно надеяться получить разложение N на простые множители.

В 1975 г. Моррисон и Бриллхарт стали перемножать сравнения (19) при различных i с тем, чтобы таким способом получить квадрат целого числа в правой части. Этот метод, метод непрерывных дробей, позволил впервые разложить на множители седьмое число Ферма $F_7 = 2^{128} + 1$. Для реализации алгоритма выбирается так называемая база множителей $\{p_1, p_2, \ldots, p_s\}$. В неё входят ограниченные по величине некоторым параметром простые числа такие, что $\left(\dfrac{N}{p_i}\right) = 1$. Последнее условие связано с тем, что, согласно (18), в разложение на простые множители чисел Q_i могут входить лишь те простые, для которых N является квадратичным вычетом.

На первом этапе алгоритма каждое очередное число Q_i делится на все числа $p_1, p_2, \ldots, p_s$ и, если оно не разлагается полностью в произведение степеней этих простых, то отбрасывается. Иначе получается разложение

$$(-1)^i Q_i = (-1)^{a_0} \prod_{j=1}^{s} p_j^{a_j}. \tag{20}$$

Этому номеру i сопоставляется вектор $(a_0, a_1, \ldots, a_s)$ (вектор показателей). Затем вычисляется следующее значение Q_{i+1}, и с ним проделывается в точности такая же процедура.

Эти вычисления проводятся до тех пор, пока не будет построено $s+2$ вектора показателей. В получившейся матрице показателей, очевидно, можно подобрать вектора-строки так, что их сумма будет вектором с чётными координатами $2(b_0, b_1, \ldots b_s)$. Если Δ — множество номеров векторов, вошедших в эту сумму, то, как легко проверить с помощью (19), имеет место сравнение

$$\left(\prod_{i\in\Delta} A_{i-1}\right)^2 \equiv \left(\prod_{j=1}^{s} p_j^{b_j}\right)^2 \pmod{N}.$$

Если с помошью этого сравнения не удаётся разложить N на множители, разложение в непрерывную дробь продолжается, продолжается набор векторов показателей и т. д.

В этот алгоритм был внесен ряд усовершенствований: вместо $\sqrt{N}$ можно раскладывать в непрерывную дробь число $\sqrt{kN}$, где маленький множитель k подбирается так, чтобы в базу множителей вошли все малые простые; была предложена так называемая стратегия раннего обрыва и т. д. Сложность этого алгоритма была оценена в 1982 г. величиной $O(\exp(\sqrt{1{,}5 \cdot \ln N \cdot \ln\ln N}))$. При выводе этой оценки использовался ряд правдоподобных, но не доказанных гипотез о распределении простых чисел. Получившаяся в оценке функция растет медленнее любой степенной функции. Алгоритмы, сложность которых оценивается подобным образом, получили название субэкспоненциальных (в зависимости от $\ln N$).

В 1982 г. Померанцем был предложен ещё один субэкспоненциальный алгоритм — алгоритм квадратичного решета. Его сложность оценивается такой же функцией, как и в методе непрерывных дробей, но вместо константы 1,5 получена лучшая — 9/8. Обозначим $m = \left[\sqrt{N}\right]$, $Q(x) = (x+m)^2 - N$ и выберем ту же базу множителей, что и в методе непрерывных дробей. При малых целых значениях x величина $Q(x)$ будет сравнительно невелика. Следующий шаг объясняет название алгоритма — квадратичное решето. Вместо того, чтобы перебирать числа x и раскладывать соответствующие значения $Q(x)$ на множители, алгоритм сразу отсеивает негодные значения x, оставляя лишь те, для которых $Q(x)$ имеет делители среди элементов базы множителей.

Задав некоторую границу B, для каждого простого числа p, входящего в базу множителей, и каждого показателя степени a, с условием $p^a \leqslant B$ находим решения x квадратичного сравнения $Q(x) \equiv 0 \pmod{p^a}$. Множество решений обозначим буквой Λ. Итак, для каждого $x \in \Lambda$ найдётся элемент базы множителей, а может быть и не один, входящий в некоторой степени в разложение на простые сомножители числа $Q(x)$. Те числа

x, при которых значения $Q(x)$ оказываются полностью разложенными, дают нам вектор показателей, как и в алгоритме непрерывных дробей. Если таких векторов окажется достаточно много, с ними можно проделать те же операции, что и в алгоритме непрерывных дробей.

Мы кратко описали здесь лишь основную идею алгоритма. Помимо этого, используется много других дополнительных соображений и различных технических приемов. Например, аналог соотношения (20) имеет вид

$$Q(x) = q_1 q_2 (-1)^{a_0} \prod_{j=1}^{s} p_j^{a_j} \pmod{N}. \tag{21}$$

В нем допускается наличие двух дополнительных больших простых множителей $B_1 < q_i < B_2$. Эти множители впоследствии при перемножении значений $Q(x)$ исключаются.

Некоторые детали реализации алгоритма можно найти в работе [6]. Отметим здесь только, что на множители раскладывалось число $5N$, база множителей состояла из -1 и 524338 простых чисел, меньших, чем $B_1 = 16333609$. При этом было использовано $B_2 = 2^{30}$. В результате просеивания получилось 112011 соотношений вида (21) без множителей q_i, 1431337 соотношений с одним таким множителем и 6881138 соотношений с двумя множителями. Именно на поиск всех этих соотношений понадобились 220 дней и большое количество работавших параллельно компьютеров. На втором шаге алгоритма, когда из соотношений (21) комбинировались чётные векторы показателей степеней, приходилось работать с матрицами, размеры которых измерялись сотнями тысяч битов. Этот второй шаг потребовал 45 часов работы. Уже четвёртый вектор с чётными показателями привёл к искомому разложению на множители.

Заключение

Мы затронули в этой статье лишь небольшую часть вопросов, связанных с теоретико-числовыми алгоритмами и оценками их сложности. За рамками остались даже чрезвычайно важные для криптографии вопросы дискретного логарифмирования, т. е. поиска чисел x, удовлетворяющих сравнению $a \equiv b^x \pmod{p}$ при заданных целых a, b, p, см. например, [20, 21]. Мы не описывали перспективные исследования, связанные с распространением алгоритмов решета на поля алгебраических чисел (решето числового поля), и использование их для разложения целых чисел на множители или решения задачи дискретного логарифмирования, см. [22, 20, 25]. Именно с помощью этих алгоритмов достигнуты теоретические

оценки сложности разложения на множители $\exp(c(\ln N)^{1/3}(\ln\ln N)^{2/3})$. Не были затронуты эллиптические кривые, т. е. определённые с точностью до обратимого множителя пропорциональности множества точек

$$E_{a,b} = \{(x, y, z) \in (\mathbb{Z}/m\mathbb{Z})^3 | y^2 z = x^3 + axz^2 + bz^3\},$$

обладающие групповой структурой. С их помощью удалось построить весьма эффективные алгоритмы разложения чисел на множители и проверки целых чисел на простоту. В отличие от мультипликативной группы $(\mathbb{Z}/m\mathbb{Z})^*$, порядок группы $E_{a,b}$ при одном и том же m меняется в зависимости от целых параметров a, b. Это оказывается весьма существенным, например, при разложении чисел m на множители. Мы отсылаем читателей за подробностями использования эллиптических кривых к статье [23].

Список литературы

[1] *Ященко В. В.* Основные понятия криптографии // Математическое просвещение. Сер. 3, №2, 1998. С. 53–70.

[2] *Rivest R. L., Shamir A., Adleman L.* A method for obtaining digital signatures and public key cryptosystems // Commun. ACM. V.21, No 2, 1978. P. 120–126.

[3] *Gardner M.* A new kind of cipher that would take millions of years to break // Sci. Amer. 1977. P. 120–124.

[4] *Виноградов И. М.* Основы теории чисел. М.: Наука, 1972.

[5] *Карацуба А. А.* Основы аналитической теории чисел. М.: Наука, 1983 г.

[6] *Atkins D., Graff M., Lenstra A. K. and Leyland P. C.* The magic words are squeamish ossifrage // ASIACRYPT–94, Lect. Notes in Comput. Sci. V. 917. Springer, 1995.

[7] *Кнут Д.* Искусство программирования на ЭВМ. Т.2: Получисленные алгоритмы. М.: Мир, 1977.

[8] *Ахо А., Хопкрофт Дж., Ульман Дж.* Построение и анализ вычислительных алгоритмов. М.: Мир, 1979.

[9] *Варновский Н. П.* Криптография и теория сложности // Математическое просвещение. Сер. 3, №2, 1998. С. 71–86.

[10] *Williams H. C.* Primality testing on a computer // Ars Combin., 5, 1978. P. 127–185. (Русский перевод: Кибернетический сборник, вып. 23, 1986. С. 51–99.)

[11] *Василенко О. Н.* Современные способы проверки простоты чисел // Кибернетический сборник, вып. 25, 1988. С. 162–188.

[12] *Alford W. R., Granville A., Pomerance C.* There are infinitely many Carmichael numbers // Ann. Math. 140, 1994. P. 703–722.

[13] *Прахар К.* Распределение простых чисел. М.: Мир, 1967.

[14] *Plaisted D. A.* Fast verification, testing, and generation of large primes // Theor. Comp. Sci. 9, 1979. P. 1–16.

[15] *Adleman L. M., Pomerance C., Rumely R. S.* On distinguishing prime numbers from composite numbers // Annals of Math. 117, 1983. P. 173–206.

[16] *Lenstra H. W. (jr.)* Primality testing algorithms (after Adleman, Rumely and Williams) // Lecture Notes in Math. V. 901, 1981. P. 243–257.

[17] *Cohen H., Lenstra H. W. (jr.)* Primality testing and Jacobi sums // Math. of Comput. V. 42, №165, 1984. P. 297–330.

[18] *Riesel H.* Prime numbers and computer methods for factorization. Birkhauser, 1985.

[19] *Cohen H.* A course in computational algebraic number theory. Graduate-Texts in Math. V. 138. New York, Springer, 1993.

[20] *Coppersmith D., Odlyzko A. M., Schroeppel R.* Discrete logarithms in $GF(p)$ // Algorithmica. V. 1, 1986. P. 1–15.

[21] *McCurley K. S.* The discrete logarithm problem // Proc. of Symp. in Appl. Math. V. 42, 1990. P. 49–74.

[22] *Lenstra A. K., Lenstra H. W., Manasse M. S., Pollard J. M.* The number field sieve // Proc. 22nd Ann. ACM Symp. on Theory of Computing. Baltimore, May 14–16, 1990. P. 564–572.

[23] *Lenstra H. W. (jr.)* Elliptic curves and number-theoretic algorithms // ICM86. P. 99–120. (Русский перевод: Международный конгресс математиков в Беркли, М.: Мир, 1991, С. 164–193.)

[24] *Koblitz N.* A Course in Number Theory and Cryptography. 2nd ed. Springer, 1994.

[25] *Lenstra A. K., Lenstra H. W. (jr.)* The Development of the Number Field Sieve. Lect. Notes in Math. V. 1554. Springer, 1993.

[26] *Ben-Or M.* Probabilistic algorithms in finite fields. Proc. 22 IEEE Symp. Found. Comp. Sci, 1981. P. 394–398.