Обозначения

$\mathbb{N}$	множество натуральных чисел: 1, 2, 3, ...
$\mathbb{Z}$	множество целых чисел: 0, ± 1, ± 2, ...
$\lvert S\rvert$	число элементов множества S
$g(n) = O(f(n))$	существуют n_0 и $C > 0$, что $g(n) \leq Cf(n)$ при $n > n_0$
НОД (a, b)	наибольший общий делитель целых чисел a и b
$\lfloor a \rfloor$	округление a вниз до целого
$a \bmod n$	остаток от деления a на n
$d \mid a$	d делит a
$a \equiv b \pmod{n}$	a сравнимо с b по модулю n, т. е. $a \bmod n = b \bmod n$
$[a]_n$	класс эквивалентности по модулю n ($\{ak + n \mid k \in \mathbb{Z}\}$)
$\langle a \rangle$	подгруппа, порожденная элементом a
$\mathbb{Z}_n$	множество классов эквивалентности по модулю n
$\mathbb{Z}_n^+$	аддитивная группа вычетов
$\mathbb{Z}_n^*$	мультипликативная группа вычетов
$\varphi(n)$	функция Эйлера, $\varphi(n) = \lvert\mathbb{Z}_n^*\rvert$
$\square$	окончание доказательства теоремы

Введение

Пособие основано на части курса лекций по криптографии, читаемого на математическом факультете Петрозаводского государственного университета студентам специальности «Информационные системы и технологии». В нем излагаются основные факты теории чисел и теории групп, необходимые для понимания принципов работы алгоритмов криптосистем с открытым ключом, таких, как RSA, протокол Диффи — Хеллмана, схема ЭльГамаля. Основными источниками при подготовке материала служили [5, 8].

В пособии приведены математические основы: делимость, простые и составные числа, множество вычетов, арифметика по модулю целого числа. Включены определения группы, подгруппы, аддитивной и мультипликативной групп вычетов, циклической группы, поля. Рассматриваются используемые в криптографии: китайская теорема об остатках, функция Эйлера и ее свойства, теорема Эйлера и малая теорема Ферма, символ Лежандра. Большинство теорем и утверждений приводится с доказательствами. Обоснование некоторых из них (не самых сложных) остается читателю в качестве упражнения.

Основную алгоритмическую часть составляют три метода: расширенный алгоритм Евклида, метод повторного возведения в квадрат и схема построения больших простых чисел. Расширенный алгоритм Евклида нахождения наибольшего общего делителя необходим для проверки взаимной простоты двух чисел и нахождения обратного элемента мультипликативной группы вычетов. Возведение в степень является “рабочей лошадкой” многих асимметричных криптосистем. Простые числа широко используются в криптографии, поэтому в пособии рассматривается один из методов построения больших простых чисел, основанный на применении теста Рабина — Миллера.

Предполагается, что читатель знаком с основными понятиями криптографии. Коротко напомним их.

Основной задачей криптографии является *шифрование* информации. При шифровании *исходное сообщение* преобразуется в *шифротекст*. Обратное действие называется *расшифрованием*. Используемые при этом алгоритмы называются *шифрами*. Для выполнения операций шифрования и расшифрования применяется секретная сменяемая часть шифра, называемая *ключом*. В *симметричных криптосистемах* для выполнения обеих операций применяется один и тот же ключ. *Дешифрованием* назы-

вается получение исходного сообщения из шифротекста без применения ключа. *Криптостойкость (надежность)* шифра характеризует его способность противостоять дешифрованию.

В *асимметричных криптосистемах* применяется пара ключей: *открытый* и *закрытый*. Открытый ключ общеизвестен, закрытый держится в тайне, и его знает только владелец ключа. Любой желающий может отправить ему сообщение, зашифрованное открытым ключом, используя при передаче обычные незащищенные каналы связи. Открытый и закрытый ключи математически связаны, однако для вычисления закрытого ключа по открытому потребуется очень большое время работы самых мощных компьютеров. Одно из важных применений асимметричной криптографии — добавление *цифровой подписи*. Цифровая подпись используется для установления подлинности сообщения.

Более подробное описание основ криптографии можно найти в [1, 3, 7, 10].

Для многих алгоритмов после их описания приводятся их реализации на псевдокоде. При этом используются распространенные в языках высокого уровня конструкции, такие, как оператор ветвления if..then..else, операторы цикла for, while, do..while, возврат из функции return. Операторные скобки (begin, end) опускаются, блок определяется отступом от края листа. Мы не делаем различия между подпрограммой, процедурой и функцией. Типы переменных определяются из контекста алгоритма. Функции могут возвращать пары, тройки и т. д. элементов. Если одна функция имеет несколько реализаций (описаний), то при программировании можно использовать любую из них.

1 Делимость и простые числа

Будем говорить, что число d *делит* число a (a *делится* на d, a *кратно* d или d — *делитель* a), если существует такое целое число k, что $a = k \cdot d$. Обозначение — $d \mid a$. Ноль кратен любому числу.

Утверждение 1.1 *Если $a \mid b$ и $b \mid c$, тогда $a \mid c$.*

Доказательство. Действительно, если $a \mid b$, то $b = an$ для некоторого целого n. Аналогично, из $b \mid c$ следует $c = bm$ для некоторого целого m. Но тогда $c = a(nm)$ и, следовательно, $a \mid c$. □

Утверждение 1.2 *Если $d \mid a$ и $d \mid b$, то $d \mid (ax + by)$ для любых целых x и y.*

Доказательство. Пусть $d \mid a$ и $d \mid b$. Тогда $a = dn$ и $b = dm$ для некоторых целых n и m. Тогда для любых целых x и y имеем $ax + by = = dnx + dmy$, то есть $ax + by = d(nx + my)$. Отсюда следует, что $d \mid (ax + by)$. □

Любое целое число, большее 1, имеет не менее двух делителей, например, 1 и самого себя. Число, не имеющее положительных делителей, кроме 1 и самого себя, называется *простым*. Целое число, большее 1, не являющееся простым, называется *составным*. Все отрицательные числа, 0 и 1 не являются ни простыми, ни составными.

Утверждение 1.3 *Если d — наименьший отличный от единицы делитель целого числа n, большего единицы, то d — простое число.*

Доказательство. Предположим, что число d составное. Тогда оно имеет некоторый делитель q, такой, что $1 < q < d$. Но тогда $q \mid d$ и $d \mid n$ и по утверждению 1.1 получаем $q \mid n$, а это противоречит тому, что d является наименьшим отличным от единицы делителем числа n. □

Докажем утверждение, известное как теорема Евклида.

Утверждение 1.4 *Существует бесконечное количество простых чисел.*

Доказательство. Предположим, что количество простых чисел конечно. Обозначим их $p_1, p_2, \ldots, p_k$. Рассмотрим число

$$n = p_1 p_2 \ldots p_k + 1.$$

Число n больше любого простого числа, и значит, составное. Пусть d — наименьший делитель числа n, больший единицы. По утверждению 1.3, число d простое. Пусть $d = p_i$ для некоторого $1 \leq i \leq k$. Тогда $p_i \mid n$ и $p_i \mid (n-1)$. Согласно свойству 1.2, имеем $p_i \mid (n - (n-1))$, т. е. $p_i \mid 1$. Но $p_i > 1$ и не может делить единицу. Полученное противоречие доказывает утверждение. $\square$

Пусть даны два целых числа: a и положительное n. Разделив a на n, мы получим:

$q = \left\lfloor \frac{a}{n} \right\rfloor$ — целое частное,

$r = a - q \cdot n = a \bmod n$ — остаток от деления.

При этом

$$a = \left\lfloor \frac{a}{n} \right\rfloor \cdot n + a \bmod n.$$

Например, для $a = 27$, $n = 4$

$q = \left\lfloor \frac{27}{4} \right\rfloor = 6$,

$r = 27 - 4 \cdot 6 = 3$.

В теории чисел всегда $0 \leq a \bmod n < n$, однако во многих языках программирования оператор вычисления остатка от деления mod работает так, что знак результата может быть отрицательным. Например, при вычислении на языке Pascal получим: $-10 \bmod 3 = -1$. В этих случаях при получении отрицательного остатка нужно прибавлять к нему делитель (в рассмотренном примере: $-1 + 3 = 2$ — верное значение).

Имеет место теорема [2]:

Теорема 1.5 *Для любого целого числа a и любого натурального n существуют единственные целые числа q и r, причем $0 \leq r < n$, для которых выполняется равенство: $a = q \cdot n + r$.* $\square$

Будем говорить, что число a *сравнимо с числом b по модулю n* ($a \equiv b \pmod{n}$), если остаток от деления a на n равен остатку от деления b на n, т. е. $a \bmod n = b \bmod n$ или, то же самое, n делит разницу $a - b$ ($n \mid (a - b)$). Запись $a \not\equiv b \pmod{n}$ означает, что a не сравнимо с b по модулю n.

Например, $27 \equiv 12 \pmod 5$, так как 27 и 12 имеют один и тот же остаток 2 от деления на 5 (число 5 делит разницу $27 - 12 = 15$).

Докажите в качестве упражнения следующее утверждение.

Утверждение 1.6 *Сравнение* $a \equiv b \pmod n$ *равносильно тому, что* $a = b + n \cdot t$ *для некоторого целого* t. $\square$

Отметим некоторые свойства сравнений, аналогичные свойствам равенств.

Утверждение 1.7 *Пусть* $a_1 \equiv b_1 \pmod n$, $a_2 \equiv b_2 \pmod n$, m — *целое число,* k — *неотрицательное целое число. Тогда:*

- $(a_1 + a_2) \equiv (b_1 + b_2) \pmod n$;
- $(a_1 + m) \equiv (b_1 + m) \pmod n$;
- $(a_1 + m \cdot n) \equiv a_1 \pmod n$;
- $(a_1 \cdot a_2) \equiv (b_1 \cdot b_2) \pmod n$;
- $(m \cdot a_1) \equiv (m \cdot b_1) \pmod n$;
- $a_1^k \equiv b_1^k \pmod n$.

Доказательство. Докажем первое из этих свойств. Согласно утверждению 1.6, имеем:

$$a_1 = b_1 + nt_1, \qquad a_2 = b_2 + nt_2$$

для некоторых целых t_1 и t_2. Тогда

$$a_1 + a_2 = b_1 + b_2 + n(t_1 + t_2)$$

и, применив утверждение 1.6, получаем:

$$(a_1 + a_2) \equiv (b_1 + b_2) \pmod n.$$

Остальные свойства доказываются аналогично. $\square$

Все целые числа можно разделить на n *классов эквивалентности по модулю* n. Класс эквивалентности, содержащий целое число a, определим так:

$$[a]_n = \{a + k \cdot n \mid k \in \mathbb{Z}\}.$$

Например, $[4]_5 = [9]_5 = [14]_5 = \{\ldots, -6, -1, 4, 9, 14, \ldots\}$. Элементы классов эквивалентности $[a]_n$ называются *вычетами по модулю* n по отношению к числам того же класса.

Обозначим через $\mathbb{Z}_n$ множество всех классов эквивалентности по модулю n:

$$\mathbb{Z}_n = \{[a]_n \mid 0 \leq a \leq n-1\}.$$

Выбрав в каждом классе по представителю, будем считать, что

$$\mathbb{Z}_n = \{0, 1, \ldots, n-1\}$$

— множество, состоящее из всевозможных остатков целых чисел от деления на n. Например:

$$\mathbb{Z}_8 = \{0, 1, 2, 3, 4, 5, 6, 7\}.$$

Введем правила выполнения арифметических операций по модулю числа n (модулярная арифметика). Для этого определим на $\mathbb{Z}_n$ операции сложения, вычитания и умножения следующим образом. Если результат соответствующей операции выходит за пределы множества $\mathbb{Z}_n$, то он заменяется остатком от деления на n. Например, в $\mathbb{Z}_5$ $4 + 3 = 2$ (берем остаток от деления 7 на 5).

2 Расширенный алгоритм Евклида

Договоримся рассматривать лишь положительные делители чисел. Любое целое число, делящее одновременно a и b, называется их *общим делителем*. Максимальный из общих делителей чисел a и b называется *наибольшим общим делителем* и обозначается НОД (a, b).

Два целых числа a и b называются взаимно простыми, если их наибольший общий делитель равен единице. Целые числа $n_1, n_2, \ldots, n_k$ называются *попарно взаимно простыми*, если любые два из них взаимно простые.

Для нахождения наибольшего общего делителя двух целых неотрицательных чисел a и b можно воспользоваться известным школьным приемом — разложить оба числа на простые множители (такое разложение будет единственным) и взять общие из них в наименьших степенях. Например, найдем НОД $(336, 90)$. Имеем:

$$336 = 2^4 \cdot 3 \cdot 7, \qquad 90 = 2 \cdot 3^5 \cdot 5.$$

Общие простые множители: 2 и 3. Перемножаем их наименьшие степени в разложении исходных чисел: $2 \cdot 3 = 6$. Значит, НОД $(336, 90) = 6$.

Однако такой подход требует большого объема вычислений при нахождении наибольшего общего делителя двух произвольных больших чисел. До сих пор не известно эффективных алгоритмов разложения чисел на множители. Более того, на вычислительной трудности этой задачи основана надежность многих криптографических систем.

Имеется несколько алгоритмов нахождения наибольшего общего делителя, применимых и для больших чисел. Самым популярным из них является алгоритм Евклида. Докажем теорему.

Теорема 2.1 *Если $a \geq 0$ и $b > 0$, то НОД $(a, b) =$ НОД $(b, a \bmod b)$.*

Доказательство. Имеем

$$a \bmod b = a - \left\lfloor \frac{a}{b} \right\rfloor b. \tag{1}$$

Если некоторое d делит a и b, то делит и любую их линейную целочисленную комбинацию, в том числе и (1), то есть d делит $a \bmod b$.

Аналогично

$$a = \left\lfloor \frac{a}{b} \right\rfloor b + a \bmod b, \tag{2}$$

и если d делит b и $a \bmod b$, то делит и a.

Выходит, что у пар $(a,\ b)$ и $(b,\ a \bmod b)$ общие делители, значит, и наибольший общий делитель у них одинаковый. □

Эта теорема является основой для алгоритма Евклида нахождения НОД (a, b). Приведем его в виде рекурсивной процедуры. На ее вход поступают неотрицательные целые числа a и b. На выходе — их наибольший общий делитель.

```
EUCLID(a, b)
  if b = 0 then
      return a
  return EUCLID(b, a mod b)
```

Предложим также итеративный вариант алгоритма:

```
EUCLID(a, b)
  while b > 0 do
      t = b
      b = a mod b
```

$a = t$
return a

Рассмотрим пример вычисления НОД $(336, 90)$. Значения переменных a и b на каждой итерации цикла представлены в таблице:

a	b	$a \bmod b$
336	90	66
90	66	24
66	24	18
24	18	6
18	6	0
6	0	—

Последнее значение переменной $a = 6$, при котором $b = 0$ и дает искомый результат.

Оценим число итераций цикла (или рекурсивных вызовов), выполняемых алгоритмом Евклида. Для этого нам понадобятся числа Фибоначчи, определяемые рекуррентным соотношением:

$$f_0 = 0, \quad f_1 = 1, \quad f_k = f_{k-1} + f_{k-2}, \text{ при } k \geq 2.$$

Рассмотрим теорему Ламе.

Теорема 2.2 *Для любого целого положительного числа k число итераций в процедуре* EUCLID(a, b) *для нахождения наибольшего общего делителя чисел a и b, $a > b \geq 0$, $b < f_k$, менее k.*

Доказательство.

Прежде докажем, что $a \geq f_{k+2}$ и $b \geq f_{k+1}$, если процедура EUCLID(a, b) выполняет k $(k \geq 1)$ итераций цикла. Для этого применим метод математической индукции.

База индукции. Рассмотрим случай, когда происходит только одна итерация цикла. Тогда $b \geq 1 = f_2$. По условию теоремы $a > b$, следовательно, $a \geq 2 = f_3$.

Индуктивное предположение. Пусть если тело цикла выполнено $k-1$ раз, то $a \geq f_{k+1}$ и $b \geq f_k$.

Доказательство индукции. Предположим, что было выполнено k итераций цикла. После первой итерации цикла значения переменных a и b заменяются соответственно на b и $a \bmod b$, затем выполняется еще

$k-1$ итераций цикла. По предположению индукции имеем $b \geq f_{k+1}$ и $a \bmod b \geq f_k$. Так как $a > b > 0$ ($b \neq 0$, так как при $b = 0$ тело цикла не выполняется ни разу), то $\lfloor a/b \rfloor \geq 1$ и

$$a = \left\lfloor \frac{a}{b} \right\rfloor \cdot b + a \bmod b \geq b + a \bmod b \geq f_{k+1} + f_k = f_{k+2}.$$

Тем самым мы установили, что если $a > b$, то для выполнения k итераций необходимо, чтобы $b \geq f_{k+1}$.

Следовательно, если $a > b \geq 0$ и $b < f_{k+1}$, то в алгоритме выполняется менее k итераций цикла. $\square$

Известно [9], что $f_k \approx \frac{1}{\sqrt{5}} \left(\frac{1+\sqrt{5}}{2} \right)^k$. Поэтому число итераций цикла алгоритма Евклида составляет $O(\log b)$.

Для дальнейших рассуждений нам понадобится приведенная ниже теорема.

Теорема 2.3 *Наибольший общий делитель двух не равных нулю одновременно целых чисел a и b является наименьшей целочисленной линейной комбинацией a и b (наименьшим положительным элементом множества $\{ax + by : \ x, y \in \mathbb{Z}\}$).*

Доказательство. Пусть s — наименьший положительный элемент множества $\{ax + by : \ x, y \in \mathbb{Z}\}$. Тогда $s = ax + by$ для некоторых целых x и y. Обозначим $q = \left\lfloor \frac{a}{s} \right\rfloor$. Имеем

$$a \bmod s = a - qs = a - q(ax + by) = a(1 - qx) + b(-qy).$$

Следовательно, $a \bmod s$ является линейной комбинацией a и b. Но s — наименьший положительный такой элемент. Значит, $a \bmod s = 0$ и s делит a. Аналогично, s делит b. Таким образом, s — общий делитель a и b и НОД $(a, b) \geq s$.

Но, с другой стороны, НОД (a, b) делит a и делит b, следовательно, делит и любую их целочисленную линейную комбинацию, в том числе и s. Так как $s > 0$, то НОД $(a, b) \leq s$.

Из этого можно сделать вывод, что НОД $(a, b) = s$. $\square$

Докажите в качестве упражнения следствия к теореме 2.3.

Следствие 2.4 *Для любых положительных целых чисел a, b, n, если $n \mid ab$ и НОД$(a, n) = 1$, то $n \mid b$.* $\square$

Следствие 2.5 *Для любых целых чисел a, b и k выполнено равенство* $НОД(a, b) = НОД(a + kb, b)$. □

Докажем две следующие теоремы.

Теорема 2.6 *Для любых целых чисел a, b, n, если* $НОД(a, n) = 1$ *и* $НОД(b, n) = 1$, *то* $НОД(ab, n) = 1$.

Доказательство. Пусть $\text{НОД}(a, n) = 1$ и $\text{НОД}(b, n) = 1$. Тогда по теореме 2.3 найдутся такие целые x_1, y_1, x_2 и y_2, для которых

$$ax_1 + ny_1 = 1 \quad \text{и} \quad bx_2 + ny_2 = 1.$$

Перемножим левые и правые части этих уравнений и сгруппируем:

$$ab(x_1x_2) + n(ax_1y_2 + bx_2y_1 + ny_1y_2) = 1.$$

Применив еще раз теорему 2.3, получим:

$$\text{НОД}(ab, n) = 1.$$

□

Теорема 2.7 *Если число p – простое и $p \mid ab$, то p делит хотя бы одно из чисел a и b.*

Доказательство. Применим метод от противного. Пусть p — простое, $p \mid ab$, при этом $\text{НОД}(a, p) = 1$ и $\text{НОД}(b, p) = 1$. Тогда по теореме 2.6 $\text{НОД}(ab, p) = 1$ и p не делит произведение ab. Получаем противоречие.

Значит, $\text{НОД}(a, p) = p$ или $\text{НОД}(b, p) = p$ (p не имеет других делителей, кроме 1 и самого себя). Следовательно, p делит или a, или b. □

Перейдем к обсуждению расширенного алгоритма Евклида. С его помощью можно получить не только наибольший общий делитель d целых неотрицательных чисел a и b, но и коэффициенты x, y, для которых

$$d = ax + by. \tag{3}$$

Такие x и y существуют по теореме 2.3. Обсудим идею алгоритма.

- Если $b = 0$, то НОД $(a, 0) = a$. Положим, $d = a$, $x = 1$, $y = 0$, при этом выполняется равенство (3).

- Рассмотрим случай $b \neq 0$. По теореме 2.1

$$d = \text{НОД}\,(a, b) = \text{НОД}\,(b, a \bmod b).$$

Пусть для некоторых целых чисел x' и y' выполнено

$$d = bx' + (a \bmod b)y'. \tag{4}$$

Перепишем равенство (4) так:

$$d = bx' + (a - \lfloor a/b \rfloor b)y' = ay' + b(x' - \lfloor a/b \rfloor y') = ax + by,$$

где $x = y'$, $y = x' - \lfloor a/b \rfloor y'$. При таких значениях x и y выполняется (3).

Приведем рекурсивную процедуру ExtendedEuclid, которая получает на входе пару чисел $a, b \geq 0$ и возвращает три искомых для них целых числа d, x, y.

```
ExtendedEuclid(a, b)
  if b = 0 then
      return (a, 1, 0)
  (d, x', y') = ExtendedEuclid(b, a mod b)
  x = y'
  y = x' − ⌊a/b⌋y'
  return (d, x, y)
```

Рассмотрим пример, когда на входе подаются $a = 336$, $b = 90$:

a	b	$a \bmod b$	$\lfloor a/b \rfloor$	d	x	y
336	90	66	3	6	-4	15
90	66	24	1	6	3	-4
66	24	18	2	6	-1	3
24	18	6	1	6	1	-1
18	6	0	3	6	0	1
6	0	—	—	6	1	0

Здесь каждая строка соответствует одному вызову рекурсивной процедуры ExtendedEuclid(a, b).

Рассмотрим циклический вариант расширенного алгоритма Евклида:
ExtendedEuclid(a, b)
  $x = 1, \quad y = 0, \quad x' = 0, \quad y' = 1$
  while $b > 0$ do
    $q = \lfloor a/b \rfloor$
    $t = b, \quad b = a \bmod b, \quad a = t$
    $t = x', \quad x' = x - q \cdot x', \quad x = t$
    $t = y', \quad y' = y - q \cdot y', \quad y = t$
  return (a, x, y)

Рассмотрим работу ExtendedEuclid при $a = 336$ и $b = 90$.

a	b	q	x	y	x'	y'
336	90	—	1	0	0	1
90	66	3	0	1	1	-3
66	24	1	1	-3	-1	4
24	18	2	-1	4	3	-11
18	6	1	3	-11	-4	15
6	0	3	-4	15	15	-56

Обратите внимание, что для каждой строки таблицы выполнено:

$$a = 336 \cdot x + 90 \cdot y, \qquad b = 336 \cdot x' + 90 \cdot y'.$$

Это и есть инварианты цикла; формулы для обновления переменных x и y подобраны так, чтобы сохранялись эти соотношения (убедитесь в этом самостоятельно). После последней итерации цикла $a = 6$, НОД $(336, 90) = 6 = 336 \cdot (-4) + 90 \cdot 15$.

Число итераций цикла (рекурсивных вызовов) процедуры ExtendedEuclid, как и прежде, равно $O(\log b)$.

В криптографии расширенный алгоритм Евклида применяется для проверки взаимной простоты двух чисел и для поиска обратного числа по модулю p.

3 Группы

В этом и следующем параграфах напомним некоторые определения, известные из курса алгебры. В частности, рассмотрим понятия группы,

полугруппы и поля.

Группой $(S, \circ)$ называется множество S с заданной на нем бинарной операцией $\circ$, подчиняющейся аксиомам:

1. **Замкнутость:** $a \circ b \in S$ для любых $a, b \in S$.
2. **Ассоциативность:** $(a \circ b) \circ c = a \circ (b \circ c)$ для любых $a, b, c \in S$.
3. **Существование нейтрального элемента:** существует такой элемент $e \in S$, что $a \circ e = e \circ a = a$ для любого $a \in S$.
4. **Существование обратного элемента:** для всякого $a \in S$ найдется элемент $b \in S$, для которого $a \circ b = b \circ a = e$.

Группа называется *абелевой*, если групповая операция $\circ$ коммутативна, т. е. $a \circ b = b \circ a$ для любых $a, b \in S$. Группа называется *конечной*, если число ее элементов конечно. Число элементов конечной группы называется ее *порядком*.

Введем в рассмотрение две группы, элементами которых являются остатки от деления на n (вычеты по модулю n). В первой из них групповой операцией будет сложение, во второй — умножение.

Аддитивная группа вычетов — группа, элементами которой являются вычеты, принадлежащие $\mathbb{Z}_n$, групповая операция — сложение по модулю n, при этом полагаем:

$$[a]_n + [b]_n = [a + b]_n.$$

Будем обозначать такую группу $(\mathbb{Z}_n, +)$ или, короче, $\mathbb{Z}_n^+$.

Для примера рассмотрим аддитивную группу $\mathbb{Z}_6^+$. В таблице для каждой пары вычетов приведены значения их суммы:

$+$	0	1	2	3	4	5
0	0	1	2	3	4	5
1	1	2	3	4	5	0
2	2	3	4	5	0	1
3	3	4	5	0	1	2
4	4	5	0	1	2	3
5	5	0	1	2	3	4

Вычитание в $\mathbb{Z}_n^+$ обратно сложению и выполняется в соответствии с правилами модулярной арифметики:

$$[a]_n - [b]_n = [a - b]_n.$$

Мультипликативная группа вычетов — группа, состоящая из вычетов множества $\mathbb{Z}_n$, взаимно простых с n. Групповой операцией является умножение по модулю n:

$$[a]_n \cdot [b]_n = [a \cdot b]_n.$$

Применяемое обозначение: $(\mathbb{Z}_n^*, \cdot)$ или, короче, $\mathbb{Z}_n^*$.

В качестве примера возьмем мультипликативную группу $\mathbb{Z}_{10}^*$ (состоит из вычетов: 1, 3, 7, 9):

$\cdot$	1	3	7	9
1	1	3	7	9
3	3	9	1	7
7	7	1	9	3
9	9	7	3	1

Покажем, как находить обратные элементы мультипликативной группы вычетов по модулю n. Пусть $a \in \mathbb{Z}_n^*$. При этом НОД $(a, n) = 1$, поэтому найдутся такие целые числа x и y, для которых $ax + ny = 1$ (теорема 2.3). Тогда $ny = 1 - ax$ и n делит разность 1 и ax, значит, $ax \equiv 1 \pmod{n}$. Таким образом, x является обратным к a элементом в $\mathbb{Z}_n^*$ (правильнее сказать, класс $[x]_n$ является обратным к $[a]_n$, но нам удобнее рассматривать отдельные представители классов).

Элемент, обратный к a относительно операции умножения по модулю n, удобно обозначать $a^{-1} \pmod{n}$ или просто a^{-1}, если понятно, о каком модуле идет речь. Частное a/b в $\mathbb{Z}_n^*$ определяется как $ab^{-1} \pmod{n}$.

Рассмотрим пример. Найдем обратный элемент к 5 в $\mathbb{Z}_{18}^*$. Для решения уравнения $1 = 5x + 18y$ применяем расширенный алгоритм Евклида. Вызов процедуры ExtendedEuclid(5, 18) вернет тройку $d = 1$, $x = -7$, $y = 2$. Так как $-7 \bmod 18 = 11$, то $5^{-1} \equiv 11 \pmod{18}$. Действительно, $(5 \cdot 11) \bmod 18 = 55 \bmod 18 = 1$.

Определим *функцию Эйлера* $\varphi(n)$, значение которой равно числу элементов группы $\mathbb{Z}_n^*$ (количеству чисел взаимно простых с n и не больших n):

$$\varphi(n) = |\mathbb{Z}_n^*|.$$

Например, $\varphi(10) = 4$, так как всего имеется 4 числа, не больших 10 и взаимно простых с 10.

Рассмотрим еще одно важное алгебраическое понятие. *Полем* называется множество S, содержащее не менее двух элементов, на котором заданы две бинарные операции: сложение $+$ и умножение $\cdot$, при этом выполняются следующие аксиомы:

1. **Замкнутость:** $a + b \in S$ и $a \cdot b \in S$ для любых $a, b \in S$.
2. **Коммутативность:** $a+b = b+a$ и $a \cdot b = b \cdot a$ для любых $a, b, c \in S$.
3. **Ассоциативность:** $(a + b) + c = a + (b + c)$ и $(a \cdot b) \cdot c = a \cdot (b \cdot c)$ для любых $a, b, c \in S$.
4. **Дистрибутивность:** $(a+b) \cdot c = (a \cdot c)+(b \cdot c)$ для любых $a, b, c \in S$.
5. **Существование нулевого элемента:** существует элемент $0 \in S$, что $a + 0 = a$ для любого $a \in S$.
6. **Существование противоположного элемента:** для всякого $a \in S$ найдется элемент $b \in S$, для которого $a + b = 0$.
7. **Существование единичного элемента:** существует элемент $1 \in S$, что $a \cdot 1 = a$ для любого $a \in S$.
8. **Существование обратного элемента:** для всякого ненулевого $a \in S$ найдется элемент $b \in S$, для которого $a \cdot b = 1$.

Таким образом, все элементы поля образуют абелеву группу по сложению (*аддитивная группа поля*), а все ненулевые элементы — абелеву группу по умножению (*мультипликативная группа поля*). *Полем Галуа* называется поле с конечным множеством элементов.

Простейший пример поля Галуа — поле вычетов $\mathbb{Z}_p$ по простому модулю p (с операциями сложения и умножения по модулю p). Именно это поле используется во многих криптосистемах с открытым ключом.

4 Подгруппы

Пусть $(S, \circ)$ — группа. Если для некоторого $S' \subset S$ $(S', \circ)$ тоже является группой, то $(S', \circ)$ называется *подгруппой* группы $(S, \circ)$.

Имеет место теорема [6]:

Теорема 4.1 *Если* $(S, \circ)$ *— конечная группа,* $S' \subset S$, $S' \neq \emptyset$, $a \circ b \in S'$ *для любых* $a, b \in S'$, *то* $(S', \circ)$ *является подгруппой группы* $(S, \circ)$. $\square$

Например, в $\mathbb{Z}_{10}^* = \{1, 3, 7, 9\}$ подгруппой является $\{1, 9\}$, т. к. для нее выполняется свойство замкнутости относительно операции умножения по модулю 10.

Число элементов подгруппы не произвольно, что подтверждается теоремой Лагранжа:

Теорема 4.2 *Если* $(S', \circ)$ *подгруппа конечной группы* $(S, \circ)$, *то* $|S'|$ *делит* $|S|$.

Доказательство. Приведем только идею доказательства (полностью его можно найти в [6]). Необходимо рассмотреть множества вида $aS' = \{a \circ x : \ x \in S'\}$, $a \in S$ (так называемые левые классы смежности по подгруппе S'). Для них несложно показать, что

$$\bigcup_{a \in S} aS' = S$$

и для произвольных a, $b \in S$ множества aS' и bS' либо совпадают, либо не пересекаются. Тогда множество S разбивается на равномощные множества вида aS'. Значит, $|S| = |S'| \cdot t$ для некоторого положительного целого t и $|S'|$ делит $|S|$. $\square$

Пусть $a \in S$. Рассмотрим степени элемента a:

$$a^{(0)} = e, \quad a^{(1)} = a, \quad a^{(2)} = a \circ a, \quad a^{(3)} = a \circ a \circ a, \ldots$$

Так как $(S,\circ)$ — конечная группа, то в последовательности степеней $a^{(0)}$, $a^{(1)}$, $a^{(2)}, \ldots$ появятся повторяющиеся элементы. Пусть $a^{(i)} = a^{(j)}$ при $i > j$. Но тогда $a^{(i-j)} = a^{(j-j)} = a^{(0)} = e$. Следовательно, существует $t > 0$, при котором $a^{(t)} = e$.

Наименьшее $t > 0$, для которого $a^{(t)} = e$, называется *порядком* элемента a и обозначается $t = \mathrm{ord}(a)$. Элементы $a^{(0)}$, $a^{(1)}, \ldots, a^{(t-1)}$ образуют подгруппу $\langle a \rangle$, *порожденную* элементом a, для которой a называется *образующим* элементом.

Например, в аддитивной группе $\mathbb{Z}_6 = \{0, 1, 2, 3, 4, 5\}$ степени элемента $a = 2$ таковы:

$$a^{(0)} = 0, \quad a^{(1)} = 2, \quad a^{(2)} = 4, \quad a^{(3)} = 0,$$

значит, $\mathrm{ord}(2) = 3$ и порождаемая подгруппа имеет вид: $\{0, 2, 4\}$.

Сказанное выше сформулируем в виде теоремы и следствия.

Теорема 4.3 *Пусть* $(S,\circ)$ *— конечная группа. Если* $a \in S$, *то* $|\langle a\rangle| = \mathrm{ord}(a)$. $\square$

Следствие 4.4 *Последовательность степеней* $a^{(1)}, a^{(2)}, \ldots$ *имеет период* $t = \mathrm{ord}(a)$. *То есть* $a^{(i)} = a^{(j)}$ *тогда и только тогда, когда* $i \equiv j \pmod{t}$. $\square$

Докажем важное следствие к теореме Лагранжа.

Следствие 4.5 *В конечной группе* $(S,\circ)$ *для любого* $a \in S$ $a^{(|S|)} = e$.

Доказательство. По теореме 4.2 $\mathrm{ord}(a)$ делит $|S|$. Следовательно, $|S| = tk$, где $t = \mathrm{ord}(a)$, k — некоторое целое число. Тогда $a^{(|S|)} = a^{tk} = (a^t)^k = e^k = e$. $\square$

Если $\mathrm{ord}(g) = |S|$, то все элементы группы $(S,\circ)$ являются степенями элемента g. В этом случае группа называется *циклической*, а g — *примитивным элементом*.

Доказательства следующих утверждений можно найти в [6] (рекомендуем доказать самостоятельно в качестве упражнения).

Утверждение 4.6 *Всякая циклическая группа абелева.* $\square$

Утверждение 4.7 *Если* $(S,\circ)$ *— циклическая группа, то для каждого делителя* d *числа* $|S|$ *в* $(S,\circ)$ *содержится ровно одна подгруппа порядка* d. $\square$

Утверждение 4.8 *Любая подгруппа циклической группы является циклической.* $\square$

Приведем теорему, для обоснования которой необходимы сведения, не рассматриваемые в этом пособии. Доказательство теоремы содержится в [7].

Теорема 4.9 *Мультипликативная группа конечного поля является циклической.* $\square$

5 Китайская теорема об остатках

Китайский математик Сун Цу около 100 г. до н.э., решая задачу о нахождении числа, которое при делении на 3, 5 и 7 дает остатки соответственно 2, 3 и 2, сформулировал теорему, которая будет изложена далее. Перед тем, как переходить к этой теореме, рассмотрим пример.

Пусть даны 2 взаимно простых числа: $n_1 = 3$ и $n_2 = 5$, $n = n_1 \cdot n_2$, и множество вычетов $\mathbb{Z}_n = \mathbb{Z}_{15} = \{0, 1 \dots 14\}$. Выпишем в таблице в левом столбце все элементы множества $\mathbb{Z}_{15}$, а в двух других — соответствующие им остатки от деления на 3 и на 5:

a	$a \bmod 3$	$a \bmod 5$
0	0	0
1	1	1
2	2	2
3	0	3
4	1	4
5	2	0
6	0	1
7	1	2

a	$a \bmod 3$	$a \bmod 5$
8	2	3
9	0	4
10	1	0
11	2	1
12	0	2
13	1	3
14	2	4

Обратим внимание, что каждая возможная комбинация остатков встречается ровно по одному разу. Выпишем в следующей таблице в первой строке все остатки от деления на 5, а в первом столбце — от деления на 3. На пересечении строк и столбцов запишем числа, дающие при делении на 3 и 5 соответствующие остатки:

	0	1	2	3	4
0	0	6	12	3	9
1	10	1	7	13	4
2	5	11	2	8	14

Получается, что мы установили взаимно однозначное соответствие между множеством $\mathbb{Z}_{15}$ и декартовым произведением $\mathbb{Z}_3 \times \mathbb{Z}_5$ множеств $\mathbb{Z}_3$ и $\mathbb{Z}_5$.

Оказывается, полученная связь удовлетворяет некоторым условиям. Сложим, для примера, 13 и 14 по модулю 15:

$$(13 + 14) \bmod 15 = 12.$$

Тот же результат можно получить так. Остатки от деления 13 на 3 и 5 равны 1 и 3. Остатки от деления 14 — 2 и 4. Сложим эти остатки между собой по соответствующим модулям:

$$(1+2) \bmod 3 = 0, \qquad (3+4) \bmod 5 = 2.$$

Паре остатков (0, 2) соответствует искомое число 12. Аналогично можно поступать при выполнении вычитания и умножения.

Перейдем теперь непосредственно к китайской теореме об остатках.

Теорема 5.1 *Пусть $n = n_1 n_2 \dots n_k$, где числа n_1, n_2,..., n_k попарно взаимно просты. Рассмотрим соответствие*

$$a \leftrightarrow (a_1, a_2, \dots, a_k), \tag{5}$$

где $a \in \mathbb{Z}_n$, $a_i \in \mathbb{Z}_{n_i}$ и $a_i \equiv a \pmod{n_i}$, $i = 1, 2, \dots, k$. Формула (5) определяет взаимно однозначное соответствие между $\mathbb{Z}_n$ и $\mathbb{Z}_{n_1} \times \mathbb{Z}_{n_2} \times \ldots \times \mathbb{Z}_{n_k}$. Причем, если

$$b \leftrightarrow (b_1, b_2, \dots, b_k),$$

то

$$(a+b) \bmod n \leftrightarrow ((a_1+b_1) \bmod n_1, (a_2+b_2) \bmod n_2, \dots, (a_k+b_k) \bmod n_k),$$

$$(a-b) \bmod n \leftrightarrow ((a_1-b_1) \bmod n_1, (a_2-b_2) \bmod n_2, \dots, (a_k-b_k) \bmod n_k),$$

$$(a \cdot b) \bmod n \leftrightarrow ((a_1 \cdot b_1) \bmod n_1, (a_2 \cdot b_2) \bmod n_2, \dots, (a_k \cdot b_k) \bmod n_k).$$

Доказательство. Докажем, что каждому элементу множества $\mathbb{Z}_n$ однозначно соответствует элемент прямого произведения множеств $\mathbb{Z}_{n_i}$. Действительно, если $a \equiv b \pmod{n}$, то n делит $a - b$. При этом n_i делит n, следовательно, n_i делит $a - b$ (утверждение 1.1) и $a \equiv b \pmod{n_i}$, $i = 1, 2, \dots, k$.

Теперь покажем, что каждому элементу прямого произведения множеств $\mathbb{Z}_{n_i}$ однозначно соответствует один элемент множества $\mathbb{Z}_n$. Обозначим $m_i = n/n_i$, $i = 1, 2, \dots, k$. Очевидно, что $m_i \equiv 0 \pmod{n_j}$ при $i \neq j$ и НОД$(m_i, n_i) = 1$ (теорема 2.6). Пусть $c_i = m_i(m_i^{-1} \bmod n_i)$, $i = 1, 2, \dots, k$. Тогда $c_i \equiv 1 \pmod{n_i}$ и $c_i \equiv 0 \pmod{n_j}$ при $i \neq j$.

Рассмотрим произвольный набор $(a_1, a_2, \ldots, a_k)$. Покажем, что ему соответствует элемент a, равный

$$a \equiv \sum_{i=1}^{k} a_i c_i \pmod{n}. \tag{6}$$

В самом деле, так как n_j — делитель n, то

$$a \equiv \sum_{i=1}^{k} a_i c_i \equiv \sum_{i \neq j} 0 \cdot a_i + 1 \cdot a_j \equiv a_j \cdot 1 \equiv a_j \pmod{n_j}, \quad j = 1, 2, \ldots, k.$$

Осталось убедиться, что такой элемент один. Это непосредственно следует из того, что число элементов множества $\mathbb{Z}_n$ совпадает с числом элементов прямого произведения множеств $\mathbb{Z}_{n_i}$. □

Следствие 5.2 *Система сравнений*

$$x \equiv a_i \pmod{n_i}, \qquad i = 1, 2, \ldots, k,$$

где $n_1, n_2, \ldots, n_k$ — попарно взаимно просты, имеет единственное решение. □

Следствие 5.3 *Система сравнений*

$$x \equiv a \pmod{n_i}, \qquad i = 1, 2, \ldots, k,$$

где числа $n_1, n_2, \ldots, n_k$ — попарно взаимно просты, эквивалентна сравнению

$$x \equiv a \pmod{n},$$

при $n = n_1 n_2 \ldots n_k$. □

Следствие 5.3 мы будем использовать при обосновании криптосистемы RSA.

Для вычисления значения a по известным $a_1, a_2 \ldots, a_k$ можно воспользоваться формулой (6). Существует и другой, более быстрый способ, для вычисления числа a [9]. Приведем его для $k = 2$ (формула Гарнера [8]).

Утверждение 5.4 *Пусть $n = n_1 n_2$, где числа n_1 и n_2 взаимно просты. Пусть $a \in \{0, 1, \ldots, n-1\}$, $a_1 = a \bmod n_1$ и $a_2 = a \bmod n_2$. Тогда*

$$a = (((a_1 - a_2)(n_2^{-1} \bmod n_1)) \bmod n_1) \cdot n_2 + a_2. \tag{7}$$

Доказательство. Покажем, что a, определенное по формуле (7), находится в диапазоне $0 \le a \le n-1$. Очевидно, что $a \ge 0$. Обозначим

$$r = ((a_1 - a_2)(n_2^{-1} \bmod n_1)) \bmod n_1.$$

Значение r является результатом вычисления по модулю n_1, значит, $r \le n_1 - 1$. Поэтому

$$r \cdot n_2 \le (n_1 - 1) \cdot n_2$$

и, в виду $a_2 \le n_2 - 1$, получаем:

$$a = r \cdot n_2 + a_2 \le (n_1 - 1) \cdot n_2 + n_2 - 1 = n_1 n_2 - 1 = n - 1.$$

Таким образом, $0 \le a \le n-1$.

Покажем теперь, что $a \bmod n_1 = a_1$ и $a \bmod n_2 = a_2$. Имеем:

$$a \bmod n_1 = ((((a_1 - a_2)(n_2^{-1} \bmod n_1)) \bmod n_1) \cdot n_2 + a_2) \bmod n_1.$$

Избавимся от одного лишнего взятия $\bmod n_1$:

$$a \bmod n_1 = ((a_1 - a_2)(n_2^{-1} \bmod n_1) \cdot n_2 + a_2) \bmod n_1.$$

Так как $((n_2^{-1} \bmod n_1) \cdot n_2) \bmod n_1 = 1$, то

$$a \bmod n_1 = ((a_1 - a_2) + a_2) \bmod n_1 = a_1 \bmod n_1 = a_1.$$

Теперь вычислим

$$a \bmod n_2 = ((((a_1 - a_2)(n_2^{-1} \bmod n_1)) \bmod n_1) \cdot n_2 + a_2) \bmod n_2.$$

Так как $(((a_1 - a_2)(n_2^{-1} \bmod n_1)) \bmod n_1) \cdot n_2$ кратно n_2, то

$$a \bmod n_2 = a_2 \bmod n_2 = a_2.$$

□

В качестве упражнения сравните число действий, необходимых для вычисления значения a при $k = 2$ по формулам (6) и (7).

6 Свойства функции Эйлера

Докажем некоторые полезные свойства функции Эйлера $\varphi(n)$.

Свойство 6.1 *Если числа m, n взаимно простые, то*

$$\varphi(mn) = \varphi(m)\varphi(n).$$

Доказательство. Пусть m и n — взаимно простые числа. Любое целое число, взаимно простое с mn, будет взаимно простым и с m, и с n.

Пусть a — положительное целое число, меньшее m и взаимно простое с ним. Рассмотрим последовательность чисел:

$$a,\ a+m,\ a+2m, \ldots, a+(n-1)m. \tag{8}$$

Докажем, что все они попарно не сравнимы по модулю n. Пусть это не так и для некоторых неравных i и j $(0 \leq i, j \leq n-1)$ $a + im \equiv$ $\equiv a + jm \pmod{n}$. Тогда n делит $im - jm = (i-j)m$. Так как n и m взаимно простые, то по следствию 2.4 n делит $i - j$ и $i \equiv j \pmod{n}$. Значит, i и j отличаются друг от друга на величину, не меньшую n, что невозможно. Тем самым все элементы последовательности (8) попарно отличаются друг от друга по модулю n.

Это означает, что последовательность (8) содержит ровно по одному представителю множества $\mathbb{Z}_n$ и совпадает с ним по числу элементов. Число элементов $\mathbb{Z}_n$, взаимно простых с n, равно (по определению) $\varphi(n)$.

Так как a взаимно простое с m, то каждый член последовательности (8) взаимно прост с m (следствие 2.5).

Всего существует $\varphi(m)$ положительных чисел, меньших m и взаимно простых с m. Порождаемые этими числами последовательности вида (8) не содержат общих элементов. Следовательно, всего существует $\varphi(m)\varphi(n)$ положительных целых чисел, меньших и взаимно простых с m и с n, а значит, и с mn (теорема 2.6).

Тем самым мы доказали, что $\varphi(mn) = \varphi(m)\varphi(n)$. □

Свойство 6.2 *Если p — простое число, k — натуральное число, то*

$$\varphi(p^k) = p^k - p^{k-1} = p^{k-1}(p-1).$$

Доказательство. Положительные целые числа не превышающие p^k и не взаимно простые с p^k имеют вид:

$$p,\ 2p,\ 3p, \ldots, p^{k-1}p.$$

Всего таких чисел p^{k-1}. Остальные целые положительные числа взаимно просты с p^k. Следовательно, $\varphi(p^k) = p^k - p^{k-1}$. $\square$

Свойство 6.3 *Если p — простое число, то $\varphi(p) = p - 1$.*

Доказательство. Следует из свойства (6.2) при $k = 1$. $\square$

Свойство 6.4 *Если $n > 2$, то $\varphi(n)$ — четное.*

Доказательство. Любое целое число $n > 2$ может быть представлено как $n = 2^k m$, где $k > 1$ и m — нечетное или как $n = p^k m$, где p — простое, взаимно простое с m.

В первом случае $\varphi(n) = \varphi(2^k m) = \varphi(2^k)\varphi(m) = 2^{k-1}\varphi(m)$ и $\varphi(n)$ — четное.

Во втором $\varphi(n) = \varphi(p^k m) = \varphi(p^k)\varphi(m) = p^{k-1}(p-1)\varphi(m)$. Так как $p - 1$ четное, то $\varphi(n)$ тоже четное. $\square$

Используя свойства функции Эйлера, мы можем вычислить значение $\varphi(n)$ для любого n. Правда, для этого нужно разложить n на множители, а для больших чисел это очень трудоемкая операция.

Рассмотрим пример. Пусть $n = 360$. Имеем:

$$360 = 2^3 \cdot 3^2 \cdot 5.$$

$$\varphi(2^3 \cdot 3^2 \cdot 5) = \varphi(2^3) \cdot \varphi(3^2) \cdot \varphi(5) =$$
$$= (2^3 - 2^2) \cdot (3^2 - 3^1) \cdot (5 - 1) = 4 \cdot 6 \cdot 4 = 96.$$

7 Мультипликативная группа вычетов

В этом параграфе будем рассматривать мультипликативную группу вычетов $\mathbb{Z}_n^*$. Докажем теорему Эйлера.

Теорема 7.1 *Если $n > 1$ — целое число, то*

$$a^{\varphi(n)} \equiv 1 \pmod{n} \tag{9}$$

для любого $a \in \mathbb{Z}_n^*$.

Доказательство. По определению функции Эйлера, $\varphi(n) = |\mathbb{Z}_n^*|$. Применив следствие 4.5 к элементам группы $\mathbb{Z}_n^*$, мы докажем сравнение (9). □

Если число n простое, то теорема Эйлера превращается в малую теорему Ферма:

Теорема 7.2 *Если p — простое число, то*

$$a^{p-1} \equiv 1 \pmod{p} \tag{10}$$

для любого $a \not\equiv 0 \pmod{p}$.

Доказательство. Так как число p простое, то $\varphi(p) = p - 1$ и группу $\mathbb{Z}_p^*$ образуют элементы, взаимно простые с p. □

Теорема 7.3 *Если p — простое число, то группа $\mathbb{Z}_p^*$ циклическая.*

Доказательство. Множество вычетов $\mathbb{Z}_p$ с операциями сложения и умножения по модулю p является полем (убедитесь в этом самостоятельно), а $\mathbb{Z}_p^*$ — его мультипликативная группа. Дальнейшее следует из теоремы 4.9. □

Следовательно, при простом p в $\mathbb{Z}_p^*$ найдется хотя бы один примитивный элемент g, степенями которого являются все остальные элементы этой группы. То есть $\mathrm{ord}(g) = |\mathbb{Z}_p^*| = p - 1$ и группа $\mathbb{Z}_p^*$ представима в виде:

$$g, g^2, \ldots, g^{p-1}.$$

Пусть g — примитивный элемент группы $\mathbb{Z}_p^*$. Тогда для любого $a \in \mathbb{Z}_p^*$ существует k такой, что $g^k \equiv a \pmod{p}$. Такое k называется *дискретным логарифмом* или *индексом* элемента a по основанию g и обозначается $\mathrm{ind}_{p,g}(a)$.

Имеет место теорема:

Теорема 7.4 *Пусть p — простое число, а g — примитивный элемент группы $\mathbb{Z}_p^*$. Тогда*

$$g^x \equiv g^y \pmod{p} \Longleftrightarrow x \equiv y \pmod{(p-1)}.$$

Доказательство. Пусть $g^x \equiv g^y \pmod{p}$. По следствию 4.4 последовательность степеней g имеет период $t = \mathrm{ord}(g) = p - 1$, следовательно, $x \equiv y \pmod{(p-1)}$.

Обратно, пусть $x \equiv y \pmod{p-1}$. Тогда для некоторого целого s $x = y + s(p-1)$. Поэтому

$$g^x \equiv g^{y+s(p-1)} \equiv g^y \cdot (g^{p-1})^s \equiv g^y \cdot 1^s \equiv g^y \pmod{p}.$$

□

Из этой теоремы следует, что величина $\mathrm{ind}_{p,g}(a)$ определена с точностью до слагаемого, кратного $p-1$.

В дальнейшем нам потребуется следующее утверждение.

Утверждение 7.5 *Для простого $p > 2$ уравнение*

$$x^2 \equiv 1 \pmod{p} \tag{11}$$

имеет только два решения в $\mathbb{Z}_p^*$: $x \equiv \pm 1 \pmod{p}$.

Доказательство. Пусть $x^2 \equiv 1 \pmod{p}$. Тогда $p \mid (x^2-1)$, следовательно, $p \mid (x-1)(x+1)$. Так как p — простое, то $p \mid (x-1)$ либо $p \mid (x+1)$ (теорема 2.7). Последнее эквивалентно тому, что $x \equiv 1 \pmod{p}$ или $x \equiv -1 \pmod{p}$. □

Корни уравнения $x^2 \equiv 1 \pmod{n}$, не сравнимые с ± 1 по модулю n, называются *нетривиальными корнями из 1 в $\mathbb{Z}_n^*$*. Очевидно, что если в $\mathbb{Z}_n^*$ имеется нетривиальный корень, то число n составное.

Пусть $p > 2$ — простое число. Элемент $a \in \mathbb{Z}_p^*$ называется *квадратичным вычетом*, если существует $x \in \mathbb{Z}_p^*$, для которого $x^2 \equiv a \pmod{p}$. В противном случае a называется *квадратичным невычетом*. Введем *символ Лежандра*, который определяется следующим образом:

$$\left(\frac{a}{p}\right) = \begin{cases} 0 & a \equiv 0 \pmod{p}, \\ 1 & a \not\equiv 0 \pmod{p} \text{ и } a \text{ — квадратичный вычет}, \\ -1 & a \not\equiv 0 \pmod{p} \text{ и } a \text{ — квадратичный невычет}. \end{cases}$$

Доказательства следующих утверждений можно найти в [9].

Утверждение 7.6 *В группе $\mathbb{Z}_p^*$ ровно половина элементов являются квадратичными вычетами, а другая половина — квадратичными невычетами.* □

Утверждение 7.7

$$\left(\frac{a}{p}\right) \equiv a^{(p-1)/2} \pmod{p}. \tag{12}$$

□

Таким образом, символ Лежандра можно использовать для определения того, является ли a квадратичным вычетом в $\mathbb{Z}_p^*$.

8 Метод повторного возведения в квадрат

Основные арифметические операции над длинными числами (сложение, вычитание, умножение, целочисленное деление, вычисление остатка от деления) могут быть реализованы обычными школьными методами (сложение в столбик, деление уголком и т. д.). При этом они будут работать относительно быстро (хотя имеются способы ускорения выполнения этих операций, но в этом пособии они не рассматриваются). Возведение же в степень по модулю некоторого числа, напротив, требует применение специального алгоритма.

Действительно, многие алгоритмы криптосистем с открытым ключом содержат операции возведения в степень по модулю целого числа, например, $a^b \bmod n$. Так как a, b, n — очень большие числа (длиной до нескольких тысяч бит), то не представляется возможным вначале подсчитать значение a^b, а затем взять его по модулю n. Во-первых, самому мощному компьютеру для этого потребуется время работы, сравнимое с возрастом Вселенной. Во-вторых, ни один компьютер не обладает количеством памяти, требуемым для хранения значения a^b. Поэтому в таком случае следует воспользоваться специальным методом вычисления $a^b \bmod n$, например, методом повторного возведения в квадрат.

Итак, пусть мы хотим вычислить $a^b \bmod n$, где a — вычет по модулю n, b — целое неотрицательное число. Идея алгоритма такова.

1. Если $b = 0$, то результатом будет 1.

2. Если $b > 0$ и является четным, то вычисляем $x = a^{b/2} \bmod n$, используя этот же алгоритм. Окончательный результат равен $x^2 \bmod n$.

3. Если $b > 0$ и является нечетным, то подсчитаем $x = a^{(b-1)/2} \bmod n$, используя этот же алгоритм. Окончательный результат равен $(a \cdot x^2) \bmod n$.

Заметим, что мы берем остатки от деления на n для всех промежуточных результатов, что не позволяет числам увеличиваться до огромных размеров.

Продемонстрируем этот метод на примере. Пусть требуется подсчитать $3^{22} \bmod 15$. Для этого нам необходимо в обратном порядке вычислить:

$$3^{11} \bmod 15, \quad 3^5 \bmod 15, \quad 3^2 \bmod 15, \quad 3^1 \bmod 15, \quad 3^0 \bmod 15.$$

Имеем:

$$\begin{aligned}
&3^0 \bmod 15 = 1,\\
&3^1 \bmod 15 = \left(3 \cdot (3^0 \bmod 15)^2\right) \bmod 15 = (3 \cdot 1^2) \bmod 15 = 3,\\
&3^2 \bmod 15 = (3^1 \bmod 15)^2 \bmod 15 = 3^2 \bmod 15 = 9,\\
&3^5 \bmod 15 = \left(3 \cdot (3^2 \bmod 15)^2\right) \bmod 15 = (3 \cdot 9^2) \bmod 15 = 3,\\
&3^{11} \bmod 15 = \left(3 \cdot (3^5 \bmod 15)^2\right) \bmod 15 = (3 \cdot 3^2) \bmod 15 = 12,\\
&3^{22} \bmod 15 = (3^{11} \bmod 15)^2 \bmod 15 = 12^2 \bmod 15 = 9.
\end{aligned}$$

Для сравнения, непосредственное вычисление 3^{22} привело бы нас к одиннадцатизначному числу 31381059609, для которого затем нужно вычислить остаток от деления на 15.

В соответствии со сказанным, напишем рекурсивную процедуру, которая по целым числам a, b, $n \geq 0$ вычисляет значение $a^b \bmod n$.

ModExp(a, b, n)
  if $b = 0$ then
    return 1
  if $b \bmod 2 = 0$ then
    $x =$ ModExp(a, $b/2$, n)
    return $x^2 \bmod n$
  $x =$ ModExp(a, $(b-1)/2$, n)
  $x = x^2 \bmod n$
  return $(a \cdot x) \bmod n$

Процедура не может работать бесконечно, поскольку рекурсивный вызов происходит с уменьшенным вдвое значением аргумента b. Таким образом, число вызовов при работе ModExp составляет $O(\log b)$.

Рекурсивный алгоритм вычисления $a^b \bmod n$ можно заменить циклическим. Для этого нам потребуется двоичная запись $\beta_k\beta_{k-1}\ldots\beta_1\beta_0$ числа b (младшие разряды справа). При этом имеем:

$$b = 2^k\beta_k + 2^{k-1}\beta_{k-1} + \ldots + 2\beta_1 + \beta_0.$$

Введем обозначение:

$$b_i = 2^{k-i}\beta_k + 2^{k-1-i}\beta_{k-1} + \ldots + 2\beta_{i+1} + \beta_i, \quad i = 0, \ldots, k.$$

Для удобства положим $b_{k+1} = 0$. Очевидно, что

$$b_{i-1} = 2^{k-i+1}\beta_k + 2^{k-i}\beta_{k-1} + \ldots + 2\beta_i + \beta_{i-1} =$$

$$= 2(2^{k-i}\beta_k + 2^{k-i-1}\beta_{k-1} + \ldots + \beta_i) + \beta_{i-1} = 2b_i + \beta_{i-1},$$

$$i = 1, \ldots, k.$$

Обозначим:

$$a_i = a^{b_i} \bmod n, \quad i = 0, \ldots, k+1.$$

В качестве окончательного результата нас интересует:

$$a_0 = a^{b_0} \bmod n = a^b \bmod n.$$

Обратим внимание, что

$$a_{k+1} = a^{b_{k+1}} \bmod n = a^0 \bmod n = 1.$$

Выведем рекуррентное соотношение для значений a_i:

$$a_{i-1} = a^{b_{i-1}} \bmod n = a^{2b_i+\beta_{i-1}} \bmod n = \left(\left(a^{b_i}\right)^2 a^{\beta_{i-1}}\right) \bmod n =$$

$$= \left(\left(a^{b_i} \bmod n\right)^2 a^{\beta_{i-1}}\right) \bmod n = \left(a_i^2 a^{\beta_{i-1}}\right) \bmod n, \ i = 1, \ldots, k+1,$$

при этом

$$a^{\beta_{i-1}} = \begin{cases} a, & \text{если } \beta_{i-1} = 1, \\ 1, & \text{если } \beta_{i-1} = 0. \end{cases}$$

Используя полученное рекуррентное соотношение, напишем процедуру вычисления значения $a^b \bmod n$:

ModExp(a, b, n)
  $k = -1$
  do
    $k = k + 1$
    $\beta_k = b \bmod 2$
    $b = \lfloor b/2 \rfloor$
  while $b > 0$

массив β содержит двоичную запись числа b
$d = 1$
for $i = k$ downto 0 do
    $d = d^2 \bmod n$
    if $\beta_i = 1$ then
        $d = (d \cdot a) \bmod n$
return d

На каждой итерации цикла for обрабатывается очередной бит двоичной записи числа b. При этом число d возводится в квадрат и, если обрабатываемый бит равен 1, умножается на a. Число шагов алгоритма по-прежнему оценивается как $O(\log b)$.

В таблице приведен пример применения циклического алгоритма для вычисления $3^{22} \bmod 15$:

i	4	3	2	1	0
β_i	1	0	1	1	0
d	3	9	3	12	9

Недостаток этого метода состоит в том, что нам необходимо предварительно построить двоичное представление числа b. Если большие целые числа в памяти компьютера представлены в двоичном виде, то делать этого не придется. Тем не менее приведем алгоритм вычисления $a^b \bmod n$, который обрабатывает биты двоичной записи числа b справа налево (от младших к старшим).

ModExp(a, b, n)
$c = a$
$d = 1$
while $b > 0$
    if $b \bmod 2 = 1$ then
        $d = (d \cdot c) \bmod n$
    $b = \lfloor b/2 \rfloor$
    $c = c^2 \bmod n$
return d

Докажите самостоятельно правильность работы этой процедуры.

9 Построение больших простых чисел

Во многих криптографических системах используются очень большие простые числа. Длина записи таких чисел в двоичной системе счисления равна 2000—4000 бит и более. Стандартный метод генерации больших простых чисел таков: взять случайное число и проверить, не будет ли оно простым. Если нет, то выбрать другое случайное число и так далее. При этом необходимо, чтобы в конечном итоге каждый раз получались простые числа, отличные от найденных ранее. Некоторые ключи шифрования формируются на основе простых чисел, а одинаковых ключей быть не должно.

Стратегия случайного выбора чисел с последующей проверкой их на простоту применима лишь в том случае, если простых чисел очень много, они встречаются относительно часто и у нас имеется эффективный алгоритм определения простоты числа. Оказывается, все это так. Простых чисел бесконечно много (утверждение 1.4), и они не так редки. При этом существует несколько хороших алгоритмов, определяющих, является ли заданное число простым или нет. Рассмотрим эти проблемы подробнее.

Начнем с вопроса о том, насколько часто простые числа встречаются среди всех положительных целых чисел. Введем функцию π, для которой $\pi(n)$ равно числу простых чисел p в интервале $[2; n]$. Например, $\pi(20) = 8$, так как имеется 8 простых чисел, не больших 20: 2, 3, 5, 7, 11, 13, 17, 19. Гаусс еще в возрасте 14 лет, рассматривая вопрос о поведении функции $\pi(n)$, заметил, что ее можно приблизить функцией $n/\ln n$. Многие другие математики занимались этим же вопросом. В результате был доказан асимптотический закон распределения простых чисел [5]. Здесь мы приведем его без доказательства.

Теорема 9.1

$$\lim_{n\to\infty} \frac{\pi(n)}{n/\ln n} = 1.$$

□

Применив этот закон, оценим вероятность P_n, с которой будет простым число q, случайно выбранное из отрезка $2 \le q \le n$. Следуя классическому определению вероятности, имеем $P_n = \frac{\pi(n)}{n}$. Так как $\pi(n) \sim \frac{n}{\ln n}$, то

$$P_n \approx \frac{n}{n \ln n} = \frac{1}{\ln n}\ .$$

Это означает, что для поиска простого числа из отрезка от 2 до n нужно перебрать примерно $\ln n$ случайных чисел. Функция $y = \ln n$ хоть и растет, но очень медленно. Так, для $n = 10^{300}$ ее значение будет равно $\ln 10^{300} = 300 \cdot \ln 10 \approx 691$. При этом количество простых чисел в интервале до 10^{300} примерно равно 10^{297}. Это намного превышает количество атомов в известной части Вселенной (10^{77}).

Остается теперь выяснить, как проверять произвольное число n на простоту. Рассмотрим утверждение.

Утверждение 9.2 *Наименьший отличный от единицы делитель d составного числа n не превосходит $\sqrt{n}$.*

Доказательство. Согласно утверждению 1.3, число d простое. При этом $n = dm$ для некоторого целого m, $1 < d \leq m < n$. Так как $m \geq d$, то $n = dm \geq d^2$ и $d \leq \sqrt{n}$. $\square$

Следовательно, можно перебрать все целые числа k из интервала от 2 до $\lfloor\sqrt{n}\rfloor$ и для каждого из них проверять делимость n на k. Если n не делится ни на одно из этих чисел, то оно простое. Однако для больших чисел n этот метод совершенно не годится. Алгоритм, основанный на этом способе, будет работать очень долго. Впрочем, такой подход можно применить для составления списка небольших простых чисел — он нам пригодится в дальнейшем.

Алгоритм генерации всех простых чисел, не превосходящих n, называется решетом Эратосфена. Он состоит в следующем.

Выпишем целые числа: $2, 3, \ldots, n$. Первое число в списке — 2, оно простое. Берем это число и вычеркиваем из списка все числа, большие двух и кратные двум. Из оставшихся чисел берем наименьшее невычеркнутое число — 3. Так как оно не вычеркнуто, то оно не делится на 2, а, следовательно, делится только на 1 и на самого себя, а потому оно тоже простое. Вычеркиваем из списка все числа, большие трех и кратные трем. Берем наименьшее из оставшихся — 5. По той же самой причине оно простое. Затем вычеркиваем все числа, большие пяти и кратные пяти. Так далее продолжаем, пока наименьшее из оставшихся чисел не превысит $\lfloor\sqrt{n}\rfloor$. Все оставшиеся невычеркнутые числа будут простыми.

Для ускорения процесса вычеркивание чисел, кратных простому p, следует начинать с p^2 (все составные числа, меньшие p^2, к этому моменту будут уже вычеркнуты).

Приведем псевдокод изложенного алгоритма. В начале работы алгоритма заполняется единицами массив $A[2..n]$. По смыслу $A[j] = 1$, если

число j простое. В процессе работы составным номерам j будут соответствовать $A[j] = 0$. Для этого в цикле while ищется очередной элемент j, для которого $A[j] = 1$. Найденное j — простое, но так как все числа i, большие j и кратные j, составные, то соответствующим элементам массива $A[i]$ будут присвоены нули. В конце работы в массив P записываются элементы j, для которых $A[j]$ остались равны единице (это и есть простые числа). Таким образом, в массив $P[1..m]$ будут записаны все простые числа, не превосходящие n.

```
ERATOSPHEN(n)
  for j = 2 to n do
        A[j] = 1
  j = 2
  while j^2 ≤ n do
        if A[j] = 1 then
              i = j^2
              while i ≤ n
                    A[i] = 0
                    i = i + j
        j = j + 1
  m = 0
  for j = 2 to n do
        if A[j] = 1 then
              m = m + 1
              P[m] = j
  return P[1..m]
```

Пример работы алгоритма для $n = 100$ приведен в таблице. При этом вычеркивались числа, кратные $2, 3, \ldots, 10$. Полученные простые числа обведены в квадраты.

	2	**3**	4	**5**	6	**7**	8	9	10
11	12	**13**	14	15	16	**17**	18	**19**	20
21	22	**23**	24	25	26	27	28	**29**	30
31	32	33	34	35	36	**37**	38	39	40
41	42	**43**	44	45	46	**47**	48	49	50
51	52	**53**	54	55	56	57	58	**59**	60
61	62	63	64	65	66	**67**	68	69	70
71	72	**73**	74	75	76	77	78	**79**	80
81	82	**83**	84	85	86	87	88	**89**	90
91	92	93	94	95	96	**97**	98	99	100

Перейдем к описанию эффективного способа проверки простоты больших чисел. Согласно малой теореме Ферма, если n — простое число и $n \geq 3$, то для любого целого числа a из интервала $2, \ldots, (n-1)$ имеем:

$$a^{n-1} \bmod n = 1. \tag{13}$$

Если для некоторого a выполнено условие (13), то число n называется *псевдопростым по основанию* a. Всякое простое число n является псевдопростым по любому основанию $a \in \mathbb{Z}_n^*$. Следовательно, если найдется такое основание $a \in 2..(n-1)$, для которого n не будет псевдопростым, то число n составное. Например, в качестве a можно взять число 2 и для него проверить выполнение (13).

Однако существует бесконечно много составных чисел, которые являются псевдопростыми по основанию 2. Например, таковым является число $341 = 11 \cdot 31$. Впрочем, эти числа встречаются не так уж часто: на 1 091 987 405 простых чисел, меньших, чем 25 000 000 000, приходится всего 21 853 составных псевдопростых по основанию 2 числа. Более того, доля составных чисел, удовлетворяющих равенству (13) для фиксированного a, стремится к нулю при $n \to \infty$.

Поэтому после проверки основания $a = 2$ можно повторить тест для $a = 3$, затем при $a = 5$ и еще нескольких значениях. Если для какого-нибудь из этих a нарушится условие (13), то проверяемое число n является составным. В противном случае можно с большой вероятностью утверждать, что оно простое (так как доля составных чисел, успешно прошедших все проведенные тесты, мала).

Такая стратегия не всегда может привести к верному ответу. Существуют составные числа n, которые являются псевдопростыми по любому

основанию a, взаимно простому с n. Они называются *псевдопростыми* числами, или *числами Кармайкла*. Числа Кармайкла достаточно редки. Их всего 16 среди первых чисел, не превосходящих 100 000; 255 до 100 000 000; 2 163 до 25 000 000 000. Первые числа Кармайкла: 561, 1 105, 1 729, 2 465, 2 821. Недавно была доказана бесконечность множества чисел Кармайкла. Также известно, что всякое число Кармайкла имеет по меньшей мере 3 простых делителя и при этом не делится на квадрат никакого простого числа (теорема Кармайкла).

Конечно, если перебирая различные основания a, случайно наткнемся на один из делителей числа Кармайкла, то мы сможем верно определить число как составное. Проблема заключается в том, что такие делители могут быть очень большими и вероятность их случайного выбора крайне мала.

Некоторая модификация предложенных тестов позволит обнаруживать и числа Кармайкла. Вычисляя $a^{n-1} \bmod n$ методом повторного возведения в квадрат, на каждой итерации алгоритма будем проверять, не встретили ли мы нетривиальный корень из единицы в $\mathbb{Z}_n^*$. Если он будет обнаружен, то число n составное (согласно утверждению 7.5, при простом n в $\mathbb{Z}_n^*$ нет нетривиальных корней из 1). На этом и основан вероятностный тест Рабина — Миллера.

Однако при неудачном выборе чисел a проверяемое число n может быть названо простым ошибочно. Доказано [1], что вероятность ложного признания составного числа в качестве простого не превосходит $1/4^r$, где r — число повторений данного теста при различных основаниях a. На практике величина r берется из интервала 3..64, подробнее см. в [5, 8].

Теперь мы можем описать шаги алгоритма построения большого простого числа n, меньшего, чем заданное N. Пусть r — количество повторений теста Рабина — Миллера.

Шаг 1. Генерируем случайным образом нечетное большое число n ($2 < n < N$).

Шаг 2. Перебираем m первых простых чисел. Если n равно одному из них, то возвращаем найденное простое n и заканчиваем алгоритм. Иначе, если n делится на одно из них, то n — составное, переходим на шаг 1. В противном случае переходим на шаг 3.

Шаг 3. Выбираем случайное число a из интервала $2..(n-1)$.

Шаг 4. С помощью алгоритма Евклида проверяем условие взаимной простоты a и n. Если оно нарушается, то число n — составное, переходим на шаг 1.

Шаг 5. Методом повторного возведения в квадрат вычисляем значение $a^{n-1} \bmod n$. При этом после каждого возведения в квадрат проверяем, не наткнулись ли мы на нетривиальный корень из 1. Если встретили, то число n — составное, переходим на шаг 1.

Шаг 6. Увеличиваем $j = j + 1$. Если $j \leq r$, то переходим на шаг 3. Иначе построено простое число n, закончим алгоритм.

Отметим, что если мы ищем очень большие числа, то на шаге 1 нужно побеспокоиться о получении чисел достаточно большого размера. На шаге 2 выполняются пробные деления на m первых простых чисел (на практике $m < 150$, список простых чисел можно получить с помощью решета Эратосфена). Тем самым отсеиваются многие составные числа. К прошедшим такое "сито" числам применяется вычислимо более тяжелый тест Рабина — Миллера.

В алгоритме использована генерация случайных чисел. Проблема получения случайных чисел требует отдельного обсуждения. Здесь мы не будем заострять на этом внимания и считаем, что у нас имеется подходящий генератор (подробности см. в [4, 8]). В псевдокоде для получения случайного числа из диапазона $m..n$ будем использовать функцию Random(m, n).

Приведем псевдокод предложенного алгоритма построения больших чисел. Функция GeneratePrime(N) реализует шаг 1, вызывает остальные шаги и возвращает простое число n. Функция IsPrime(n) проверяет заданное число n методом пробных делений (шаг 2), вызывает тест Рабина — Миллера и возвращает "истину", если число n простое, и "ложь", если составное. Функция RabinMiller (n, r) реализует основную часть алгоритма (шаги с 3 по 6) и возвращает "истину", если проверяемое число, по ее мнению, оказалось простым.

```
GeneratePrime(N)
  do
      n = Random(2, ⌊N/2⌋)
      n = 2 · n − 1
  while not IsPrime(n)
```

```
    return n

IsPrime(n)
    Используем массив простых чисел P[1..m]
    for j = 1 to m do
        if n mod P[j] = 0 then
            if n = P[j] then
                return true
            else
                return false
    r = число возможных значений a
    return RabinMiller(n, r)

RabinMiller(n, r)
    b = n − 1
    k = −1
    do
        k = k + 1
        β_k = b mod 2
        b = ⌊b/2⌋
    while b > 0
    for j = 1 to r do
        a=Random(2, n − 1)
        if Euclid(a, n) > 1 then
            return false
        d = 1
        for i = k downto 0 do
            x = d
            d = d^2 mod n
            if d = 1 and x ≠ 1 and x ≠ n − 1 then
                return false
            if β_i = 1 then
                d = (d · a) mod n
        if d ≠ 1 then
            return false
    return true
```

В предложенном псевдокоде вместо сравнения d с -1 по модулю n

мы сравниваем d с $(n-1)$ (так как $-1 \equiv (n-1) \pmod{n}$).

Известны и другие способы проверки чисел на простоту [9], но метод Рабина — Миллера является одним из самым популярных и быстрых. В дальнейшем генерацию случайных больших простых чисел мы будем использовать в нескольких криптографических алгоритмах.

10 Криптосистема Диффи—Хеллмана

Наиболее известный протокол открытого распространения ключей был разработан и опубликован в 1976 году Уитфилдом Диффи (Whitfield Diffie) и Мартином Хеллманом (Martin Hellman). Протокол позволяет двум пользователям совместно получить секретный ключ, используя при этом незащищенные каналы связи.

Для реализации метода необходимы два параметра:

p — большое простое число;

g — примитивный элемент, порождающий группу $\mathbb{Z}_p^*$.

Числа p и g открыты и могут использоваться одновременно несколькими пользователями. Они должны быть получены заранее и разосланы всем абонентам (или опубликованы).

Если участники протокола хотят создать общий секретный ключ k, то они должны выполнить следующие шаги:

1. Пользователь A выбирает случайное целое число $x \in \mathbb{Z}_p^*$ и посылает пользователю B значение $X = g^x \bmod p$.

2. Пользователь B выбирает случайное целое число $y \in \mathbb{Z}_p^*$ и посылает пользователю A значение $Y = g^y \bmod p$.

3. Пользователь A вычисляет $k = Y^x \bmod p$.

4. Пользователь B вычисляет $k' = X^y \bmod p$.

Если все сделано правильно, то $k = k' = g^{xy} \bmod p$.

Надежность протокола обусловлена трудностью задачи дискретного логарифмирования. Функция $X = g^x \bmod p$ является *односторонней функцией*: существует быстрый алгоритм вычисления X по известному x, для обратного действия (определения x по X) эффективных алгоритмов до сих пор не найдено. Также не известно быстрых методов поиска

$g^{xy} \bmod p$ по имеющимся значениям $g^x \bmod p$ и $g^y \bmod p$ (при условии, что x и y неизвестны).

В [8] приводятся рекомендации относительно размера числа p. Так, для защиты данных на 20 лет следует использовать простое число длиной 2 048 бит, на 35 лет — 3 072 бита, на 50 лет — 4 096 бит. В настоящее время считается, что длина в 6 800 бит удовлетворяет самым строгим требованиям безопасности. Но такой размер p сильно снижает производительность системы.

На самом деле параметр g может и не быть примитивным элементом группы $\mathbb{Z}_p^*$. Главное, чтобы g порождало подгруппу достаточно большого порядка. Рассмотрим этот вопрос подробнее.

Напомним, мультипликативная группа вычетов $\mathbb{Z}_p^*$ состоит из элементов:

$$1, 2, \ldots, p-1.$$

Согласно утверждению 7.3, группа $\mathbb{Z}_p^*$ циклическая и в ней найдется порождающий ее примитивный элемент. Рассмотрим подгруппы группы $\mathbb{Z}_p^*$. По утверждению 4.8, любая такая подгруппа тоже циклическая, то есть представима в виде:

$$1, a, a^2, \ldots, a^{q-1},$$

где $a \in \mathbb{Z}_p^*$ — порождающий подгруппу элемент, имеющий порядок q. При этом q является делителем $p-1$ (теорема 4.2). Обратно, для любого делителя q числа $p-1$ существует единственная подгруппа группы $\mathbb{Z}_p^*$, имеющая порядок q (утверждение 4.7).

Так как число $(p-1)$ четное, то в $\mathbb{Z}_p^*$ обязательно присутствуют подгруппы порядка 1 и 2. Первая состоит из единственного элемента: 1. Вторая включает два элемента: 1 и $p-1$. Потребуем, чтобы в $\mathbb{Z}_p^*$ не было других подгрупп малого размера.

Для этого возьмем $p = 2q + 1$, где q — простое число. Тогда $|\mathbb{Z}_p^*| = 2q$ и мультипликативная группа вычетов $\mathbb{Z}_p^*$ имеет следующие 4 подгруппы:

1. Подгруппа, состоящая из одного элемента: 1.
2. Подгруппа, состоящая из двух элементов: 1, $p-1$.
3. Подгруппа, состоящая из q элементов.
4. Подгруппа, состоящая из $2q$ элементов, совпадающая с $\mathbb{Z}_p^*$.

Получается, что каждый элемент группы $\mathbb{Z}_p^*$ порождает одну из описанных выше подгрупп. При этом первую из них порождает 1, а вторую — $p-1$. Элементы, отличные от 1 и $p-1$, порождают третью и четвертую подгруппы.

Как известно, в $\mathbb{Z}_p^*$ одна половина элементов является квадратичными вычетами, другая — квадратичными невычетами (утверждение 7.6). Так как степень любого квадратичного вычета является также квадратичным вычетом, то, очевидно, любой примитивный элемент $\mathbb{Z}_p^*$ является квадратичным невычетом. Верно и обратное: любой квадратичный невычет, отличный от $p-1$, порождает $\mathbb{Z}_p^*$. Остается заметить, что подгруппу порядка q составляют квадратичные вычеты и ее порождает любой квадратичный вычет, отличный от 1 (докажите).

Пусть мы в качестве g взяли квадратичный невычет, отличный от $p-1$. В этом случае есть одна проблема: все четные степени g будут квадратичными вычетами, а нечетные нет. Злоумышленник, зная значение X и используя формулу (12), сможет легко определить, является ли X квадратичным вычетом или нет, тем самым выяснить четность секретного x.

Поэтому g должно быть квадратичным вычетом, отличным от 1. При этом g будет порождать подгруппу порядка q.

Для генерации g можно взять случайное число α из интервала $(1; p-1)$ и вычислить:

$$g = \alpha^2 \bmod p.$$

Благодаря такому выбору, число g не будет равно 1, так как p — простое (утверждение 7.5) и не будет равно и $p-1$, так как $p-1$ — квадратичный невычет (докажите почему), а g — квадратичный вычет.

Однако при применении простых чисел вида $p = 2q + 1$ операция возведения в степень по модулю p будет выполняться очень медленно. На практике для ускорения вычислений используют подгруппы гораздо меньшего размера, чем $p-1$, но все же достаточно большие для обеспечения надежности системы.

Покажем, как это делается. Выберем в качестве q 256-битовое простое число (по современным оценкам такой размер считается безопасным). Затем найдем большое простое число p, равное $nq+1$, для некоторого четного n. Для этого число n подбирается случайно до тех пор, пока p не будет простым. Размер числа n определяется необходимым размером p (см. выше).

После этого перейдем к поиску элемента порядка q. Поступим так: выберем случайное число из $\mathbb{Z}_p^*$ и возведем его в степень n по модулю p:

$$g = \alpha^n \bmod p.$$

Если $g = 1$, то выберем еще раз α и повторно вычислим g. Легко показать, что найденное значение g будет иметь порядок q. Действительно, так как $g^q \equiv \alpha^{nq} \equiv 1 \pmod{p}$, то $\mathrm{ord}(g) \leq q$. При этом такое g порождает подгруппу, размер которой является делителем q. Но так как q — простое и $g > 1$, то $\mathrm{ord}(g) = q$.

Для определения $g^x \bmod p$ можно вычислить $g^{x \bmod q} \bmod p$ (следствие 4.4). Тем самым время выполнения операции сокращается примерно в 8 раз [8].

Остается рассмотреть критерий проверки того, что некоторое число g действительно порождает подгруппу порядка q (при заданном p). Этот критерий необходим для того, чтобы пользователи системы, не участвующие в генерации параметров, могли быть уверены в ее надежности. Для проверки параметров системы необходимо проделать следующее:

1. Убедиться в простоте и достаточной длине чисел p и q.
2. Выяснить, является ли q делителем $p - 1$.
3. Проверить выполнение условий: $g \neq 1$ и $g^q = 1 \pmod{p}$.

Если все эти три условия выполняются, то параметры p, q и g можно использовать в протоколе.

Приведем псевдокод генерации параметра g, здесь N — максимальное значение n:

```
GENERATE-G(N)
  q = GENERATEPRIME(2^256)
  do
      n = RANDOM(2, N)
      p = n · q + 1
  while not ISPRIME(p)
  do
      a = RANDOM(2, p − 1)
      g = MODEXP(a, n, p)
  while g = 1
  return (g, q, p)
```

Функция вычисления $g^x \bmod p$:
DiffieHellman(g, x, q, p)
  $x' = x \bmod q$
  X = ModExp(g, x', p)
  return X

В реальных системах протоколы (например, SSH [1]), основанные на схеме Диффи — Хеллмана, включают процедуры аутентификации пользователей. Заметим, что размер совместно полученного ключа k (несколько тысяч бит) слишком большой для применения его в симметричных алгоритмах (124–256 бит). Поэтому для окончательного получения ключа вычисляют значение хэш-функции от k.

11 Криптосистема RSA

Криптографическая система с открытым ключом RSA была предложена Рональдом Ривестом (Ronald Rivest), Ади Шамиром (Adi Shamir) и Леонардом Адлеманом (Leonard Adleman) в 1977 году и названа по первым буквам имен ее создателей. Она применяется как для шифрования данных, так и для цифровой подписи.

Приведем основные шаги алгоритма RSA.

1. Выбираем случайным образом два больших простых числа p и q, не равных друг другу.
2. Определяем криптомодуль $n = pq$.
3. Подсчитываем значение функции Эйлера $\varphi(n) = (p-1)(q-1)$.
4. Подбираем небольшое нечетное число e, взаимно простое с $\varphi(n)$, т. е. НОД $(e, \varphi(n)) = 1$.
5. Вычисляем $d = e^{-1} \bmod \varphi(n)$.
6. Составляем открытый ключ — пару (e, n).
7. Составляем закрытый ключ — пару (d, n).

Шифрование сообщения M (M — целое число в интервале от 0 до $n-1$) делается так:

$$C = E(M) = M^e \bmod n. \tag{14}$$

Для расшифрования шифротекста C нужно выполнить:

$$M = D(C) = C^d \bmod n. \tag{15}$$

Для того, чтобы к сообщению M добавить цифровую подпись, нужно возвести M в степень d по модулю n:

$$\sigma = D(M) = M^d \bmod n. \tag{16}$$

Проверка цифровой подписи σ сообщения M:

$$M' = E(\sigma) = \sigma^e \bmod n. \tag{17}$$

Если $M = M'$, то подпись верна. Таким образом, проверка цифровой подписи соответствует шифрованию открытым ключом, добавление подписи к сообщению — расшифрованию закрытым ключом.

Поясним некоторые аспекты алгоритма.

- Так как p и q взаимно простые, то значение функции Эйлера по утверждениям 6.1 и 6.3 вычисляется по формуле: $\varphi(n) = \varphi(pq) = = (p-1)(q-1)$.

- Согласно утверждению 6.4, $\varphi(n)$ — четное (впрочем, для $\varphi(n)$, равному $(p-1)(q-1)$, это очевидно и так). Чтобы число e было взаимно простым с $\varphi(n)$, необходима нечетность e.

Для обоснования правильности работы алгоритма RSA необходимо показать, что формулы (14) и (15) задают взаимно обратные перестановки на множестве вычетов $\mathbb{Z}_n$. То есть нужно доказать, что $D(E(M)) = M$ для любого $M \in \mathbb{Z}_n$.

Очень просто можно убедиться в справедливости этого утверждения почти для всех M, используя теорему Эйлера (теорема 7.1). Действительно, так как $ed \equiv 1 \pmod{\varphi(n)}$, то существует такое целое k, что

$$ed = 1 + k\varphi(n).$$

По теореме Эйлера имеем:

$$C^d \equiv (M^e)^d \equiv M^{ed} \equiv M^{1+k\varphi(n)} \equiv$$

$$\equiv M \cdot (M^{\varphi(n)})^k \equiv M \cdot 1^k \equiv M \pmod{n}$$

для всех M, взаимно простых с n. При этом доля неотрицательных чисел $M < n$, для которых НОД $(M, n) > 1$, ничтожно мала.

Впрочем, можно доказать, что сравнение $C^d \equiv M \pmod{n}$ будет выполняться абсолютно для всех $M \in \mathbb{Z}_n$.

Выразив $\varphi(n)$ через p и q, мы получим:

$$ed = 1 + k(p-1)(q-1).$$

Если $M \not\equiv 0 \pmod{p}$, то, согласно малой теореме Ферма (теорема 7.2), имеем:

$$C^d \equiv (M^e)^d \equiv M^{ed} \equiv M^{1+k(p-1)(q-1)} \equiv$$

$$\equiv M \cdot (M^{p-1})^{q-1} \equiv M \cdot 1^{q-1} \equiv M \pmod{p}.$$

Если $M \equiv 0 \pmod{p}$, то, очевидно, $M^{ed} \equiv M \pmod{p}$ и, следовательно,

$$C^d \equiv M \pmod{p} \tag{18}$$

для всех M. Аналогично доказывается, что для любого M выполнено:

$$C^d \equiv M \pmod{q}. \tag{19}$$

Ввиду того, что $n = pq$, по следствию 5.3 к китайской теореме об остатках получаем:

$$C^d \equiv M \pmod{n} \tag{20}$$

для всех $M \in \mathbb{Z}_n$.

Надежность криптосистемы RSA основана на трудности задачи разложения составного числа на множители. Действительно, для расшифрования нам нужно знать показатель d. Чтобы найти d по известному e, необходимо значение функции Эйлера $\varphi(n)$. Для вычисления $\varphi(n)$ нам понадобится разложение числа n на множители p и q. Поэтому, если мы знаем разложение числа n на множители, то легко сможем найти сообщение по заданному шифротексту (из этого следует, что числа p и q нужно держать в тайне).

До сих пор не известно эффективных методов разложения больших чисел на множители. Если кто-нибудь когда-нибудь научится быстро решать задачу разложения числа на множители, то ему удастся легко взломать систему RSA. Впрочем, до сих пор не доказано, что нет другого способа взлома этой системы (но и не найдено). Резюмируя сказанное, можно сделать вывод, что отсутствие эффективного алгоритма разложения числа на множители гораздо полезнее, чем его существование.

Отметим принципиальное отличие алгоритма RSA от алгоритма Диффи — Хеллмана. В криптосистеме Диффи — Хеллмана используется односторонняя функция $X = g^x \bmod p$, и для дешифрования злоумышленнику нужно найти дискретный логарифм $\mathrm{ind}_{p,g}(X)$, что очень сложно. В RSA шифрование осуществляется возведением в степень $C = M^e \bmod n$. Дешифрование эквивалентно поиску $C^{1/e} \bmod n$. Проблема состоит в том, что злоумышленник не знает, по какому модулю нужно вычислять обратный элемент $1/e$ (в данном случае по модулю $\varphi(n)$). Знание разложения числа n на множители делает эту задачу простой. Поэтому функция шифрования RSA принадлежит к классу *односторонних функций с лазейкой* [1]. Знание "лазейки" делает операцию вычисления ее обратной функции простой, незнание — сложной. "Лазейкой" являются числа p и q.

По современным оценкам [8], размер числа n должен составлять примерно 2 048 бит для обеспечения защиты данных на протяжении 20 лет. При возможности рекомендуется использовать n длиной около 4 096 бит. В перспективе могут потребоваться и более длинные модули, например 8 192 бит, и к этому нужно быть готовым.

Важное замечание: если число e мало, то размер шифруемого сообщения M должен быть достаточно большим, чтобы при вычислении $M^e \bmod n$ были взятия по модулю n, т. е. необходимо условие $M^e > n$.

Рассмотрим процедуру генерации пары ключей. Вначале случайным образом параметрам p и q присваиваются значения больших простых неравных друг другу чисел. Затем нужно подобрать e, взаимно простое с $(p-1)(q-1)$. Здесь мы будем последовательно перебирать нечетные числа, начиная с тройки, пока не получим требуемое значение. Параметр N задает максимальное значение чисел p и q.

GenerateKeyRSA(N)
  do
    p = GeneratePrime(N)
    q = GeneratePrime(N)

while $p = q$
$f = (p - 1) \cdot (q - 1)$
$e = 1$
do
    $e = e + 2$
    $(t, x, y) = \text{ExtendedEuclid}(e, f)$
while $t > 1$
$n = p \cdot q$
$d = x \bmod f$
return (e, d, n)

Пара (e, n) образует открытый ключ, пара (d, n) — закрытый ключ криптосистемы RSA.

Для ускорения выполнения операций шифрования и проверки цифровой подписи желательно, чтобы число e было мало и содержало как можно меньше единиц в двоичном представлении (для возведения в степень мы используем метод повторного возведения в квадрат, в нем число умножений зависит от длины показателя и числа единиц в его двоичной записи). Например, в качестве e можно взять числа: 3, 5, 17, 65 537. Приведем псевдокод генерации ключей, когда вначале задается значение e, а затем подбираются p и q:

$\text{GenerateKeyRSA}(N, e)$
do
    $p = \text{GeneratePrime}(N)$
    $q = \text{GeneratePrime}(N)$
    $f = (p - 1) \cdot (q - 1)$
    $(t, x, y) = \text{ExtendedEuclid}(e, f)$
while $p = q$ or $t > 1$
$n = p \cdot q$
$d = x \bmod f$
return (e, d, n)

Для шифрования сообщения M необходимо использовать функцию возведения в степень:

$C = \text{ModExp}(M, e, n)$.

Расшифрование выполняется аналогично, но с другим ключом:

$M = \text{ModExp}(C, d, n)$.

Для уменьшения времени выполнения расшифрования следует вос-

пользоваться китайской теоремой об остатках. Можно вначале вычислить $M_1 = C^d \bmod p$ и $M_2 = C^d \bmod q$, а затем по формуле (7) и $C^d \bmod n$. При вычислении по модулю p мы можем сократить показатель d, взяв остаток от деления его на $p-1$ (следствие 4.4). Аналогично и с вычислением по модулю q. Таким образом, получаем:

$$M_1 = C^{d \bmod (p-1)} \bmod p, \qquad M_2 = C^{d \bmod (q-1)} \bmod q,$$

$$M = (((M_1 - M_2)(q^{-1} \bmod p)) \bmod p) \cdot q + M_2.$$

При этом функция расшифрования будет выглядеть так:
DecryptRSA(C, d, p, q, n)
$d_1 = d \bmod (p-1)$
$d_2 = d \bmod (q-1)$
M_1 = ModExp(C, d_1, p)
M_2 = ModExp(C, d_2, q)
(t, x, y) = ExtendedEuclid(q, p)
$r = x \bmod p$
$M = (((M_1 - M_2) \cdot r) \bmod p) \cdot q + M_2$
return M

Отметим, что такой метод вычисления $C^d \bmod n$ позволяет сократить объем работы в 3–4 раза [8]. Остается заметить, что значение r можно было подсчитать предварительно и использовать в дальнейшем при расшифровании.

Алгоритмы асимметричной криптографии, подобные RSA, работают гораздо медленнее симметричных шифров. Поэтому на практике систему RSA применяют для передачи ключа шифрования симметричного криптоалгоритма. Для добавления цифровой подписи, как правило, подписывают не само сообщение, а его хэш-образ.

12 Криптосистема ЭльГамаля

Криптосистему, предложенную в 1985 году Тахиром ЭльГамалем (Taher ElGamal), можно использовать как для цифровых подписей, так и для шифрования.

Вначале рассмотрим генерацию пары ключей.

1. Выбираем простое число p и два случайных числа g и x, меньших p.

2. Вычисляем
$$y = g^x \bmod p.$$

3. Составляем открытый ключ — тройку (y, g, p). Числа p и g могут быть общими для группы пользователей. Параметр y должен быть уникален для каждого пользователя.

4. Закрытым ключом является число x.

Для подписания сообщения M нужно выполнить следующие шаги:

1. Выбираем случайное число k, взаимно простое с $p - 1$.

2. Вычисляем
$$a = g^k \bmod p.$$

3. Расширенным алгоритмом Евклида находим b из уравнения:
$$M \equiv (xa + kb) \pmod{(p-1)}.$$

4. Подписью является пара чисел (a, b).

Чтобы убедиться, что пара (a, b) является подписью сообщения M, необходимо проверить, что
$$y^a a^b \bmod p = g^M \bmod p.$$

Действительно,
$$y^a a^b \equiv (g^x)^a \cdot (g^k)^b \equiv g^{xa+kb} \equiv g^M \pmod{p}.$$

Число k должно храниться в секрете (лучше сразу после подписания его уничтожить). Если злоумышленник раскроет k, то он сможет раскрыть секретный ключ x. Каждое новое сообщение должно подписываться новым значением k. Если злоумышленник сумеет получить два сообщения, подписанных при помощи одного и того же k, то он сможет раскрыть x, даже не зная k.

Теперь рассмотрим, как выполняются операции шифрования и расшифрования.

Для шифрования сообщения M нужно выполнить следующее:

1. Выбираем случайное число k, взаимно простое с $p-1$.

2. Вычисляем
$$a = g^k \bmod p,$$
$$b = y^k M \bmod p.$$

3. Шифротекстом является пара чисел $(a,\, b)$.

Расшифрование происходит следующим образом:

$$M = \frac{b}{a^x} \bmod p.$$

Покажем, что расшифрование обратно шифрованию. В самом деле,

$$a^x \equiv g^{kx} \pmod p,$$
$$\frac{b}{a^x} \equiv \frac{y^k M}{a^x} \equiv \frac{g^{xk} M}{g^{kx}} \equiv M \pmod p.$$

Надежность криптосистемы ЭльГамаля определяется сложностью задачи вычисления дискретного логарифма.

Запишем рассмотренные алгоритмы на псевдокоде. Начнем с функции генерации ключей:

GenerateKeyElGamal(N)
  p = GeneratePrime(N)
  g = Random(2, $p-2$)
  x = Random(2, $p-2$)
  y = ModExp(g, x, p)
  return (p, g, y, x)

Подпись сообщения M:

Signature(M, p, g, x)
  do
    k = Random(2, $p-2$)
    (t, u, v) = ExtendedEuclid(k, $p-1$)
  while $t > 1$
  a = ModExp(g, k, p)
  $b = ((M - x \cdot a) \cdot u) \bmod (p-1)$
  return (a, b)

Проверка подписи:
VerifySignature(M, a, b, p, g, y)
  u = ModExp(y, a, p)
  v = ModExp(a, b, p)
  $w = (u \cdot v) \bmod p$
  s = ModExp(g, M, p)
  if $w = s$ then
    return **true**
  return **false**

Шифрования сообщения M открытым ключом:
EncryptElGamal(M, p, g, y)
  do
    k = Random(2, $p - 2$)
    t = Euclid(k, $p - 1$)
  while $t > 1$
  a = ModExp(g, k, p)
  b = ModExp(y, k, p)
  $b = (b \cdot M) \bmod p$
  return (a, b)

Расшифрование пары (a, b) закрытым ключом:
DecryptElGamal(a, b, x)
  s = ModExp(a, x, p)
  (t, u, v) = ExtendedEuclid(s, p)
  $M = (b \cdot u) \bmod p$
  return M

При выборе параметров p и g можно воспользоваться приемами, описанными в параграфе 10.

Стандарты цифровой подписи России ГОСТ Р 34.10–94 и США DSS используют схемы, подобные описанной выше. Обобщение алгоритма цифровой подписи данного вида можно найти в [10]. Заметим, что на практике цифровую подпись применяют не к самому сообщению, а к его хэш-образу.

Заключение

Для лучшего усвоения материала рекомендуется реализовать в виде программ на языке высокого уровня рассмотренные здесь алгоритмы и протестировать их.

В пособии остались не освещенными многие вопросы. Так, сведения из теории чисел и теории групп приведены в минимально необходимом объеме. В некоторых случаях формулируются лишь частные случаи (теорема 7.3, утверждение 5.4, определение индекса и пр.). Подробное изложение этого материала можно найти в [2, 6, 7].

Технические детали реализации алгоритмов асимметричных криптосистем и советы по их применению содержатся в [8, 10]. Там же приводятся описания многочисленных вариаций рассмотренных алгоритмов, других протоколов и схем. Подробное изложение нескольких алгоритмов построения больших простых чисел приведено в пособии [9].

Много интересной и полезной информации, посвященной криптографии, включая описание программ, научные статьи, можно найти в Интернете. Например, можно порекомендовать сайт pgpru.com проекта «PGP в России», посвященного информационной безопасности, криптографии и сетевой анонимности.

Список литературы

[1] Введение в криптографию / под общ. ред. В. В. Ященко. — М. : МЦНМО : «ЧеРо», 1998.

[2] Виноградов И. М. Основы теории чисел / И. М. Виноградов. — СПб. : Лань, 2004.

[3] Исагулиев К. П. Справочник по криптологии / К. П. Исагулиев. — Минск : Новое знание, 2004.

[4] Кнут Д. Э. Искусство программирования для ЭВМ: в 3 т. — Получисленные алгоритмы / Д. Э. Кнут. — М. : Издательский дом «Вильямс», 2005. — Т. 2.

[5] Кормен Т. Алгоритмы: построение и анализ / Т. Кормен, Ч. Лейзерсон, Р. Ривест. — М. : МЦНМО : БИНОМ. Лаборатория знаний, 2004.

[6] Курош А. Г. Теория групп: учебник / А. Г. Курош. — СПб. : Лань, 2005.

[7] Нечаев В. И. Элементы криптографии (Основы теории защиты информации) / В. И. Нечаев. — М. : Высшая школа, 1999.

[8] Фергюсон Н. Практическая криптография / Н. Фергюсон, Б. Шнайер. — М. : Издательский дом «Вильямс», 2005.

[9] Черемушкин А. В. Лекции по арифметическим алгоритмам в криптографии / А. В. Черемушкин. — М. : МЦНМО, 2002.

[10] Шнайер Б. Прикладная криптография: протоколы, алгоритмы, исходные тексты на языке Си / Б. Шнайер. — 2-е изд. — М. : Триумф, 2002.