

23 Реализация RSA

23.1 Арифметика больших чисел

Для построения на базе функции RSA криптосистемы с открытым ключом и системы ЭЦП мы должны уметь:

выполнять	зачем	алгоритм	сложность (время)
сложение $a + b$	базовая операция	столбиком	$O(\log a + \log b)$
вычитание $a - b$	базовая операция	столбиком	$O(\log a + \log b)$
умножение ab	базовая операция	столбиком	$O(\log a \log b)$
деление $a = qb + r$	базовая операция	уголком	$O(\log q \log b)$
генерация простых	p, q	следующая лекция	
вычисление (a, b)	$(e, \varphi(n)) \stackrel{?}{=} 1$	Евклида	?
обращение $b^{-1} \bmod a$	$d = e^{-1} \bmod \varphi(n)$	расш. Евклида	?
возведение в степень $a^b \bmod n$	$x^e \bmod n, y^d \bmod n$	бинарные методы	?

Систему перечисленных операций над числами, разрядность которых намного превосходит разрядность современных процессоров, принято называть *арифметикой больших чисел* (АБЧ). АБЧ встроена в пакеты компьютерной алгебры **Mathematica** и **Maple**, реализована в виде отдельных библиотек **NTL**, **Miracle**, **LiDIA** и др.

Всюду ранее мы предполагали, что алгоритмы АБЧ являются полиномиальными. Наша задача — доказать, что это действительно так.

В АБЧ числа представляются в системе счисления по основанию $B = 2^w$, где w — разрядность процессора. Это значит, что число a записывается в виде $(a_{\ell-1} \dots a_1 a_0)_B$, где $a_i \in \{0, 1, \dots, B-1\}$ и $\sum_{i=0}^{\ell-1} a_i B^i = a$.

Сложение больших чисел можно выполнить с помощью следующего алгоритма.

АЛГОРИТМ СЛОЖЕНИЕ «СТОЛБИКОМ»

Вход: $a = (a_{\ell-1} \dots a_1 a_0)_B, b = (b_{\ell-1} \dots b_1 b_0)_B$.

Выход: $c = a + b, c = (c_{\ell} \dots c_1 c_0)_B$.

Шаги:

1. $carry \leftarrow 0$.
2. Для $i = 0, \dots, \ell - 1$:
 - (1) $c_i \leftarrow (a_i + b_i + carry) \bmod B$;
 - (2) если $a_i + b_i + carry \geq B$, то $carry \leftarrow 1$, иначе $carry \leftarrow 0$.
3. $c_{\ell} \leftarrow carry$.
4. Возвратить $(c_{\ell} \dots c_1 c_0)_B$.

Сложность: требуется выполнить $\lceil \log_B a \rceil + \lceil \log_B b \rceil$ сложений чисел из $\{0, 1, \dots, B-1\}$ (формула работает даже если складываются числа разной длины). Последние сложения также можно выполнить «столбиком», используя представление в системе счисления по основанию 2. Именно так организованы сложения в ядрах процессоров.

Упражнение 23.1. Выписать алгоритмы вычитания, умножения и деления больших чисел и подтвердить оценки их сложности. □

23.2 Алгоритм Евклида

Для вычисления н.о.д. (a, b) используется алгоритм Евклида, который был открыт 22 века назад.

АЛГОРИТМ ЕВКЛИДА

Вход: натуральные a и b , $a \geq b$.

Выход: (a, b) .

Шаги:

1. Пока $b \neq 0$ выполнить:

$$(1) \ r \leftarrow a \bmod b, \ a \leftarrow b, \ b \leftarrow r.$$

2. Возвратить a .

Корректность алгоритма следует из двух фактов:

1) $(a, b) = (b, a \bmod b)$ (в цикле алгоритма мы меняем пару чисел a и b на пару b и $a \bmod b$);

2) если $a \bmod b = 0$, то $(a, b) = b$ (ср. с условием выхода из цикла и возвратом).

Проанализируем сложность алгоритма. Следующая теорема показывает, что алгоритм Евклида является полиномиальным.

Теорема 23.1 (Ламе, 1844). Цикл в алгоритме Евклида выполняется не более $\lceil \log_\phi a \rceil$ раз, где $\phi = (1 + \sqrt{5})/2$ — положительный корень уравнения $x^2 = x + 1$.

Доказательство. Пусть $r_{-1} = a$, $r_0 = b$ и r_1, r_2, \dots — последовательные значения переменной r на шагах алгоритма, т. е.

$$r_i = r_{i-2} \bmod r_{i-1} \quad \text{или} \quad r_{i-2} = r_{i-1}q_i + r_i, \quad i = 1, 2, \dots$$

Если цикл выполняется $k + 1$ раз, то $r_k \neq 0$ и $r_{k+1} = 0$. Оценим k . Для этого составим следующую таблицу:

r_{k+1}	r_k	$r_{k-1} \geq r_k + r_{k+1}$	\dots	$r_{-1} \geq r_0 + r_1$
\parallel	\vee	\vee	\dots	\vee
$s_0 = 0$	$s_1 = 1$	$s_2 = s_1 + s_0$	\dots	$s_{k+2} = s_k + s_{k+1}$

Последовательность s_0, s_1, s_2, \dots в нижней строке таблицы — это последовательность Фибоначчи (л.р.п. второго порядка над кольцом \mathbb{Z} , см. § 12.3), для элементов которой справедлива оценка

$$s_{i+2} \geq \phi^i, \quad i \geq 0.$$

Действительно, неравенство верно при $i = 0, 1$. Если неравенство верно для s_i, s_{i+1} , то

$$s_{i+2} = s_i + s_{i+1} \geq \phi^{i-2} + \phi^{i-1} = \phi^{i-2}(\phi + 1) = \phi^{i-2}\phi^2 = \phi^i$$

и обоснован шаг индукции.

Соберем полученные оценки:

$$a = r_{-1} \geq s_{k+2} \geq \phi^k \Rightarrow k \leq \log_\phi a,$$

откуда следует требуемый результат. □

Пример 23.1 (мини RSA). Пусть $p = 11$, $q = 3$. Следовательно, $n = 33$ и $\varphi(n) = 20$. Для $e = 3$ выполняется

$$(e, p-1) = (3, 10) = (3, 1) = 1, \quad (e, q-1) = (3, 2) = (2, 1) = 1 \quad \Rightarrow \quad (e, \varphi(n)) = 1,$$

т. е. $e = 3$ является допустимой открытой экспонентой. □

23.3 Расширенный алгоритм Евклида

Пусть число a обратимо по модулю b . Для нахождения $a^{-1} \pmod{b}$ можно воспользоваться теоремой Эйлера: $a^{-1} \equiv a^{\varphi(b)-1} \pmod{b}$. Другой, более эффективный, способ обращения состоит в использовании расширенного алгоритма Евклида.

АЛГОРИТМ РАСШИРЕННЫЙ ЕВКЛИДА

Вход: натуральные a, b , $a \geq b$.

Выход: (d, x, y) , где $d = (a, b)$, x, y — целые, удовлетворяющие равенству $ax + by = d$.

Комментарий: если $d = 1$, то $xa \equiv 1 \pmod{b} \Rightarrow x \equiv a^{-1} \pmod{b}$.

Шаги:

1. $r_{-1} \leftarrow a, r_0 \leftarrow b$.
2. $x_{-1} \leftarrow 1, y_{-1} \leftarrow 0, x_0 \leftarrow 0, y_0 \leftarrow 1$.
3. $i \leftarrow 0$.
4. Пока $r_i \neq 0$ выполнить:
 - (1) $i \leftarrow i + 1$;
 - (2) $q_i \leftarrow \lfloor r_{i-2}/r_{i-1} \rfloor, r_i \leftarrow r_{i-2} - q_i r_{i-1}$;
 - (3) $x_i \leftarrow x_{i-2} - q_i x_{i-1}$;
 - (4) $y_i \leftarrow y_{i-2} - q_i y_{i-1}$.
5. Возвратить $(r_{i-1}, x_{i-1}, y_{i-1})$.

Корректность алгоритма следует из того, что после каждого прохода цикла выполняется

$$x_i a + y_i b = r_i.$$

Действительно, снова применим индукцию. Равенство очевидно при $i = -1, i = 0$. Для $i \geq 1$ выполняется

$$x_i a + y_i b = (x_{i-2} - q_i x_{i-1})a + (y_{i-2} - q_i y_{i-1})b = (x_{i-2}a + y_{i-2}b) - q_i(x_{i-1}a + y_{i-1}b) = r_{i-2} - q_i r_{i-1} = r_i.$$

Расширенный алгоритм отличается от обычного только тем, что на его шагах дополнительно определяются значения x_i, y_i . Ясно, что сложность расширенного алгоритма незначительно отличается от сложности обычного.

Упражнение 23.2. Оптимизировать алгоритм, сократив количество переменных. □

Пример 23.2 (мини RSA — продолжение). Продолжим предыдущий пример и найдем $d = e^{-1} \pmod{\varphi(n)}$. Имеем: $a = \varphi(n) = 20, b = 3$,

$$(r_{-1}, x_{-1}, y_{-1}) = (20, 1, 0), (r_0, x_0, y_0) = (3, 0, 1), (r_1, x_1, y_1) = (2, 1, -6), (r_2, x_2, y_2) = (1, -1, 7), r_3 = 0.$$

Таким образом, $-1 \cdot 20 + 7 \cdot 3 = 1$ и $d = 7$. □

23.4 Возведение в степень

Для возведения a в степень b по модулю n используются бинарные методы. Суть их состоит в следующем: степень b представляется двоичным словом $b_{\ell-1} \dots b_1 b_0$ таким, что $b_{\ell-1} \neq 0$ и $b = \sum_{i=0}^{\ell-1} b_i 2^i$.

Выражение a^b может быть записано двумя способами:

$$a^b = (a)^{b_0} (a^2)^{b_1} \dots (a^{2^{\ell-1}})^{b_{\ell-1}}, \quad a^b = \left(\dots \left((a^{b_{\ell-1}})^2 a^{b_{\ell-2}} \right)^2 \dots \right)^2 a^{b_0}.$$

Каждый из этих способов определяет свой алгоритм возведения в степень:

<i>Справа налево</i>	<i>Слева направо</i>
1. Установить $u \leftarrow 1, v \leftarrow a$.	1. Установить $u \leftarrow 1$.
2. Для $i = 0, 1, \dots, \ell - 1$ выполнить:	2. Для $i = \ell - 1, \ell - 2, \dots, 0$ выполнить:
(1) если $b_i \neq 0$, то $u \leftarrow u \cdot v \bmod n$;	(1) $u \leftarrow u \cdot u \bmod n$;
(2) $v \leftarrow v \cdot v \bmod n$.	(2) если $b_i \neq 0$, то $u \leftarrow u \cdot a \bmod n$.
3. Вернуть u .	3. Вернуть u .

Для возведения в степень потребуется $\ell = \lceil \log_2 b \rceil$ возведений в квадрат и $w(b) \leq \ell$ умножений, т. е. мы снова получаем полиномиальный алгоритм.

От одного возведения в степень можно отказаться.

Пример 23.3 (мини RSA — продолжение). Пусть $x = 5$. Выполним зашифрование (справа налево):

$$y = x^3 \bmod n = x^{(11)} \bmod n = (x \bmod n)(x^2 \bmod n) \bmod n = 5 \cdot 25 \bmod 33 = 26.$$

Выполним расшифрование (слева направо):

$$y^d \bmod n = y^{(111)} \bmod n = ((y^2 \bmod n)y \bmod n)^2 y \bmod n = \dots = 5.$$

23.5 Китайская система сравнений

Определение 23.1. Пусть m_1, \dots, m_k — попарно взаимно простые числа. Следующая система уравнений называется *китайской системой сравнений*:

$$\begin{cases} x \equiv a_1 \pmod{m_1}, \\ \dots\dots\dots \\ x \equiv a_k \pmod{m_k}. \end{cases}$$

Теорема 23.2. Китайская система имеет единственное решение по модулю $M = m_1 \dots m_k$:

$$x^* = \sum_{i=1}^k M_i a_i b_i \bmod M, \quad M_i = M/m_i, \quad b_i = M_i^{-1} \bmod m_i.$$

Доказательство. Во-первых, покажем, что x^* действительно является решением. Рассмотрим вычет $x^* \bmod m_i$. В сумме $\sum_{i=1}^k M_i a_i b_i$ все слагаемые, кроме $M_i a_i b_i$, кратны m_i . Поэтому

$$x^* \equiv M_i a_i b_i \equiv a_i \pmod{m_i}.$$

Отображение $\varphi: (a_1, \dots, a_k) \mapsto x^*$ действует на $\{0, 1, \dots, M-1\}$ и является инъективным (иначе некоторое x^* являлось бы решением сразу двух систем). Количество наборов (a_1, \dots, a_k) совпадает с M . Поэтому φ — биекция и всякое $x^* \in \{0, 1, \dots, M-1\}$ является решением одной и только одной системы (a_1, \dots, a_k) . \square

Способ решения китайской системы, указанный в теореме, называется *методом Гаусса*. С помощью этого метода решение можно найти за полиномиальное время — требуется выполнить $(k-1) + 2k$ умножений, k делений, $(k-1)$ сложений и k обращений по модулю.

23.6 Оптимизация RSA

Для обеспечения надлежащей стойкости модуль RSA должен быть достаточно большим. При этом зашифрование и расшифрование по RSA выполняется на несколько порядков медленнее, чем зашифрование с помощью симметричных криптосистем. Мы рассмотрим три оптимизации, которые ускоряют RSA.

Функция Кармайкла. Функцию Эйлера можно заменить на функцию Кармайкла:

- 1) $\lambda(p^\alpha) = \varphi(p^\alpha)$, если $p \neq 2$ или $p = 2$ и $\alpha \leq 2$;
- 2) $\lambda(2^\alpha) = \frac{1}{2}\varphi(2^\alpha)$, $\alpha \geq 3$;
- 3) $(a, b) = 1 \Rightarrow \lambda(ab) = [\lambda(a), \lambda(b)]$.

Действительно, рассуждения теоремы остаются справедливыми с учетом теоремы Кармайкла: $(a, m) = 1 \Rightarrow a^{\lambda(m)} \equiv 1 \pmod{m}$.

При переходе к функции Кармайкла вместо секретной экспоненты $d = e^{-1} \pmod{\varphi(n)}$ можно использовать экспоненту $d = e^{-1} \pmod{\lambda(n)}$. Эта экспонента может быть меньше, и тогда расшифрование будет выполняться быстрее.

Пример 23.4. $\lambda(33) = [10, 2] = 10$, $d = 3^{-1} \pmod{10} = 7$ — экспонента не изменилась. □

Малая открытая экспонента. Для ускорения вычисления $x^e \pmod{n}$ используют небольшую открытую экспоненту e в двоичном разложении которой содержится малое число 1. Кроме этого, требуют, чтобы экспонента была простой (для увеличения вероятности взаимной простоты с $\varphi(n)$).

Распространенный выбор: $e = 3$ (стандарты PHEM, PKCS#1), $e = 17$, $e = 65537$ (X.509, PKCS#1). Во всех этих случаях экспонента является простым Ферма вида $2^{2^k} + 1$.

Сохранение p и q . Для ускорения вычисления $y^d \pmod{n}$ Алиса кроме своего личного ключа d сохраняет также простые p и q , а также числа $q^* = q^{-1} \pmod{p}$, $p^* = p^{-1} \pmod{q}$.

При этом вычисление $x = y^d \pmod{n}$ можно провести следующим образом:

- 1) вычислить $x_1 = y^{d \pmod{p-1}} \pmod{p}$;
- 2) вычислить $x_2 = y^{d \pmod{q-1}} \pmod{q}$;
- 3) решить китайскую систему

$$\begin{cases} x \equiv x_1 \pmod{p}, \\ x \equiv x_2 \pmod{q} \end{cases}$$

следующим образом:

$$x = (q^*qx_1 + p^*px_2) \pmod{n}.$$

Можно не сохранять p^* , а определить решение следующим образом:

$$x = (x_2 + q(q^*(x_1 - x_2) \pmod{p})) \pmod{n}.$$

Действительно, в этом случае

$$x \equiv x_2 \pmod{q}, \quad x \equiv x_2 + (x_1 - x_2) \equiv x_1 \pmod{p}.$$

Пример 23.5 (мини RSA — окончание). Вычислим $q^* = 3^{-1} \pmod{11} = 4$. Теперь для расшифрования 26 мы вычисляем

$$x_1 = 26^{7 \pmod{10}} \pmod{11} = 5, \quad x_2 = 26^{7 \pmod{2}} \pmod{3} = 2$$

и

$$x \equiv 2 + 3(4 \cdot 3 \pmod{11}) \equiv 2 + 3 \equiv 5 \pmod{33}.$$