Лабораторная работа №7

Математические основы защиты информации и информационной безопасности

Николаев Дмитрий Иванович, НПМмд-02-24

Содержание

Список иллюстраций

Список таблиц

1 Цель работы

Изучить работу алгоритмов дискретного логарифмирования в конечном поле — ρ-метод Полларда для дискретного логарифмирования, а также реализовать его программно.

2 Теоретическое введение

2.1 Дискретное логарифмирование в конечном поле

Задача дискретного логарифмирования, как и задача разложения на множители, применяется во многих алгоритмах криптографии с открытым ключом. Предложенная в 1976 году У. Диффи и М. Хеллманом для установления сеансового ключа, эта задача послужила основой для создания протоколов шифрования и цифровой подписи, доказательств с нулевым разглашением и других криптографических протоколов.

Пусть над некоторым множеством Ω произвольной природы определены операции сложения «+» и умножения «·». Множество Ω называется *кольцом*, если выполняются следующие условия:

1. Сложение коммутативно: $a + b = b + a$ для любых $a, b \in \Omega$;
2. Сложение ассоциативно: $(a + b) + c = a + (b + c)$ для любых $a, b, c \in \Omega$;
3. Существует нулевой элемент $0 \in \Omega$, такой, что $a + 0 = a$ для любого $a \in \Omega$;
4. Для каждого элемента $a \in \Omega$ существует противоположный элемент $-a \in \Omega$, такой, что $(-a) + a = 0$;
5. Умножение дистрибутивно относительно сложения:

$$a \cdot (b + c) = a \cdot b + a \cdot c, \quad (a + b) \cdot c = a \cdot c + b \cdot c,$$

для любых $a, b, c \in \Omega$.

Если в кольце Ω умножение коммутативно: $a \cdot b = b \cdot a$ для любых $a, b \in \Omega$, то кольцо называется *коммутативным*.

Если в кольце Ω умножение ассоциативно: $(a \cdot b) \cdot c = a \cdot (b \cdot c)$ для любых $a, b, c \in \Omega$, то кольцо называется *ассоциативным*.

Если в кольце Ω существует единичный элемент e такой, что $a \cdot e = e \cdot a = a$ для любого $a \in \Omega$, то кольцо называется кольцом с единицей (или *унитарным*).

Если в ассоциативном, коммутативном кольце Ω с единицей (АКУ-кольце) для каждого ненулевого элемента a существует обратный элемент $a^{-1} \in \Omega$, такой, что $a^{-1} \cdot a = a \cdot a^{-1} = e$, то кольцо называется *полем*.

Пусть $m \in \mathbb{N}$, $m > 1$. Целые числа a и b называются *сравнимыми по модулю* m (обозначается $a \equiv b \mod m$), если разность $a - b$ делится на m. Некоторые свойства отношения сравнимости:

1. *Рефлексивность*: $a \equiv a \mod m$.
2. *Симметричность*: если $a \equiv b \mod m$, то $b = a \mod m$.
3. *Транзитивность*: если $a \equiv b \mod m$ и $b = c \mod m$, то $a = c \mod m$.

Отношение, обладающее свойством рефлексивности, симметричности и транзитивности, называется отношением *эквивалентности*. Отношение сравнимости является отношением эквивалентности на множестве $\mathbb{Z}$ целых чисел.

Отношение эквивалентности разбивает множество, на котором оно определено, на *классы эквивалентности*. Любые два класса эквивалентности либо не пересекаются, либо совпадают.

Классы эквивалентности, определяемые отношением сравнимости, называются *классами вычетов по модулю* m. Класс вычетов, содержащий число a, обозначается $a \mod m$ или $\overline{a}$ и представляет собой множество чисел вида $a + km$, где $k \in \mathbb{Z}$; число a называется представителем этого класса вычетов.

Множество классов вычетов по модулю m обозначается $\mathbb{Z}/m\mathbb{Z}$, состоит ровно из m элементов и относительно операций сложения и умножения является *кольцом классов вычетов* по модулю m.

Пример. Если $m = 2$, то $\mathbb{Z}/2\mathbb{Z} = \{0 \mod 2, 1 \mod 2\}$, где $0 \mod 2 = 2\mathbb{Z}$ - множество всех четных чисел, $1 \mod 2 = 2\mathbb{Z} + 1$ - множество всех нечетных чисел.

Обозначим $F_p = \mathbb{Z}/p\mathbb{Z}$, где p - простое целое число, и назовем конечным полем из p элементов. Задача дискретного логарифмирования в конечном поле F_p формулируется так: для данных целых чисел a и b, $a > 1$, $b < p$, найти логарифм - такое целое число x, что $a^x \equiv b \mod p$ (если такое число существует). По аналогии с вещественными числами используется обозначение $x = \log_a b$.

Безопасность соответствующих криптосистем основана на том, что, зная числа a, x, p, вычислить $a^x \mod p$ легко, а решить задачу дискретного логарифмирования трудно. Рассмотрим ρ-Метод Полларда, который можно применить и для задач дискретного логарифмирования. При этом случайное отображение f должно обладать не только сжимающими свойствами, но и вычислимостью логарифма (логарифм числа $f(c)$ можно выразить через неизвестный логарифм x и $\log_a f(c)$). Для дискретного логарифмирования в качестве случайного отображения f чаще всего используются ветвящиеся отображения, например:

$$f(c) = \begin{cases} ac, & \text{при } c < \frac{p}{2}, \\ bc, & \text{при } c \geq \frac{p}{2}. \end{cases}$$

При $c < \frac{p}{2}$ имеем $\log_a f(c) = \log_a c + 1$, а при $c \geq \frac{p}{2}$ имеем $\log_a f(c) = \log_a c + x$

2.2 Алгоритм, реализующий ρ-Метод Полларда для задач дискретного логарифмирования

Вход. Простое число p, число a порядка r по модулю p, целое число b, $1 < b < p$; отображение f, обладающее сжимающими свойствами и сохраняющее вычислимость логарифма.

Выход. Показатель x, для которого $a^x = b \mod p$, если такой показатель существует.

1. Выбрать произвольные целые числа u, v и положить $c \leftarrow a^u b^v \mod p$, $d \leftarrow c$.
2. Выполнять $c \leftarrow f(c) \mod p, d \leftarrow f(f(d)) \mod p$, вычисляя при этом логарифмы для c и d как линейные функции от x по модулю r, до получения равенства $c \equiv d \mod p$.
3. Приравняв логарифмы для c и d, вычислить логарифм x решением сравнения по модулю r. Результат: x или “Решений нет”.

2.2.1 Пример

Решим задачу дискретного логарифмирования $10^x = 64 \mod 107$, используя ρ-Метод Полларда. Порядок числа 10 по модулю 107 равен 53.

Выберем отображение $f(c) \equiv 10^c \mod 107$ при $c < 53$, $f(c) \equiv 64^c \mod 107$ при $c \geq 53$. Пусть $u = 2$, $v = 2$. Результаты вычислений запишем в таблицу:

Номер шага	c	$\log_a c$	d	$\log_a d$
0	4	$2+2x$	4	$2+2x$
1	40	$3+2x$	76	$4+2x$
2	79	$4+2x$	56	$5+3x$
3	27	$4+3x$	75	$5+5x$
4	56	$5+3x$	3	$5+7x$
5	53	$5+4x$	86	$7+7x$
6	75	$5+5x$	42	$8+8x$
7	92	$5+6x$	23	$9+9x$
8	3	$5+7x$	53	$11+9x$
9	30	$6+7x$	92	$11+11x$
10	86	$7+7x$	30	$12+12x$
11	47	$7+8x$	47	$13+13x$

Таблица 2.1: Таблица шагов ρ-метода Полларда для дискретного логарифмирования

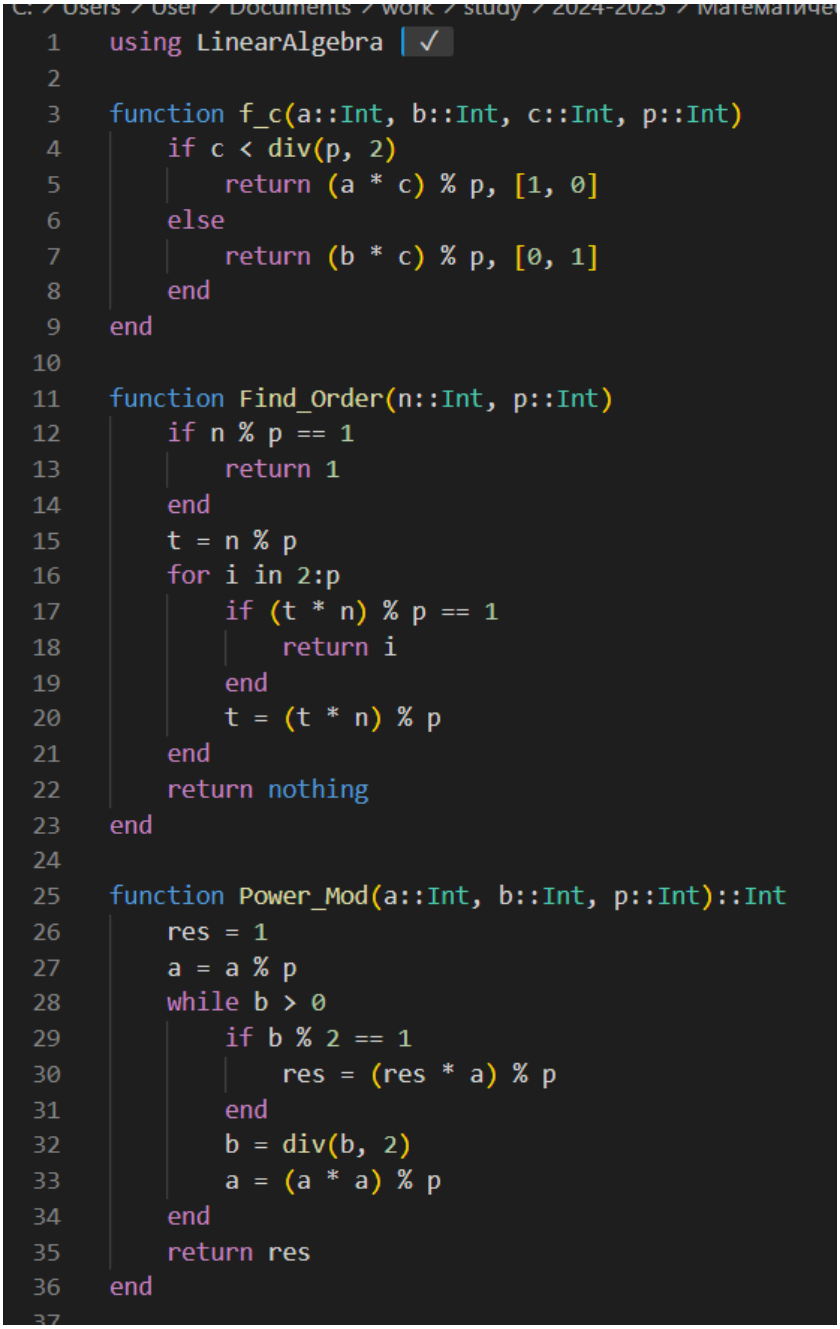
Приравниваем логарифмы, полученные на 11-м шаге: $7 + 8x = 13 + 13x \mod 53$. Решая сравнение первой степени, получаем: $x = 20 \mod 53$.

Проверка: $10^{20} = 64 \mod 107$.

3 Выполнение лабораторной работы

Действуя согласно [1], реализуем все описанные алгоритмы на языке Julia.

Сначала реализуем несколько функций (Рис.[3.1]): функцию, обладающую сжимающими свойствами; функцию по нахождению порядка числа в конечном поле, и функцию модульного экспоненциирования. Далее реализуем функцию ρ-метода Полларда для нахождения дискретного логарифма (Рис.[3.2]). После чего найдём дискретный логарифм для двух случаев: $10^{64} = x \mod 107$, $2^{22} = x \mod 29$ (Рис.[3.3]); в результате чего получим следующий вывод, представленный на Рис.[3.4].



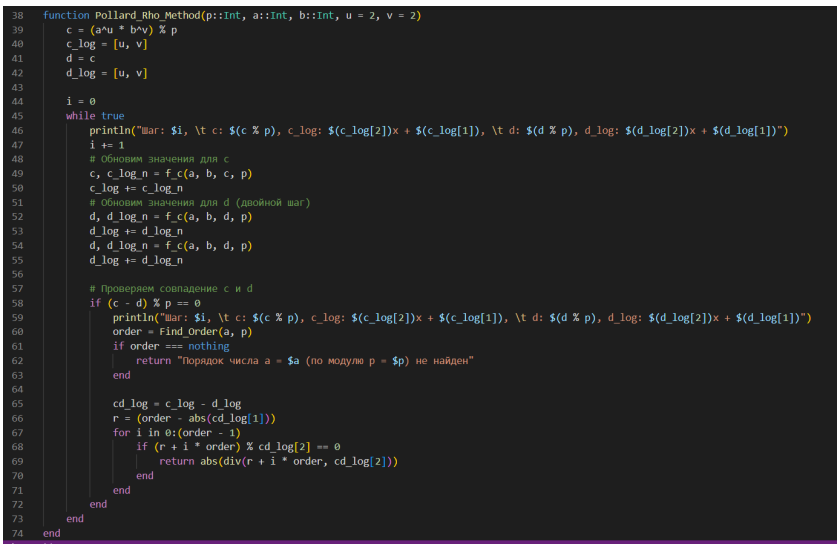
```julia
using LinearAlgebra

function f_c(a::Int, b::Int, c::Int, p::Int)
    if c < div(p, 2)
        return (a * c) % p, [1, 0]
    else
        return (b * c) % p, [0, 1]
    end
end

function Find_Order(n::Int, p::Int)
    if n % p == 1
        return 1
    end
    t = n % p
    for i in 2:p
        if (t * n) % p == 1
            return i
        end
        t = (t * n) % p
    end
    return nothing
end

function Power_Mod(a::Int, b::Int, p::Int)::Int
    res = 1
    a = a % p
    while b > 0
        if b % 2 == 1
            res = (res * a) % p
        end
        b = div(b, 2)
        a = (a * a) % p
    end
    return res
end
```

Рис. 3.1: Код вспомогательных функций на Julia



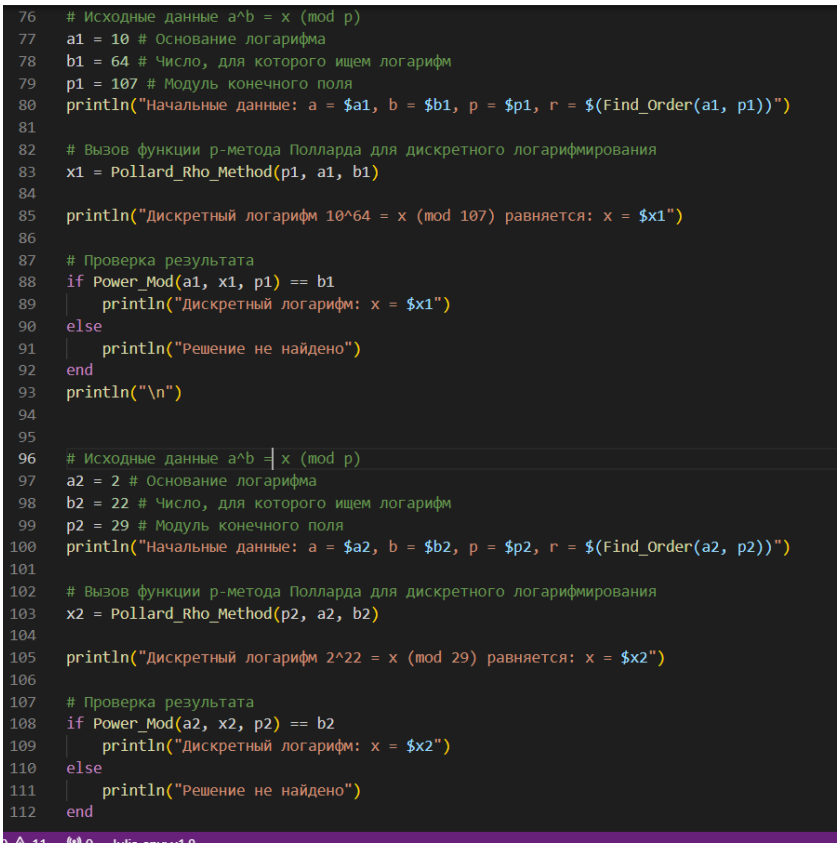
```julia
function Pollard_Rho_Method(p::Int, a::Int, b::Int, u = 2, v = 2)
    c = (a^u * b^v) % p
    c_log = [u, v]
    d = c
    d_log = [u, v]

    i = 0
    while true
        println("Шаг: $i, \t c: $(c % p), c_log: $(c_log[2])x + $(c_log[1]), \t d: $(d % p), d_log: $(d_log[2])x + $(d_log[1])")
        i += 1
        # Обновим значения для c
        c, c_log_n = f_c(a, b, c, p)
        c_log += c_log_n
        # Обновим значения для d (двойной шаг)
        d, d_log_n = f_c(a, b, d, p)
        d_log += d_log_n
        d, d_log_n = f_c(a, b, d, p)
        d_log += d_log_n

        # Проверяем совпадение c и d
        if (c - d) % p == 0
            println("Шаг: $i, \t c: $(c % p), c_log: $(c_log[2])x + $(c_log[1]), \t d: $(d % p), d_log: $(d_log[2])x + $(d_log[1])")
            order = Find_Order(a, p)
            if order === nothing
                return "Порядок числа a = $a (по модулю p = $p) не найден"
            end

            cd_log = c_log - d_log
            r = (order - abs(cd_log[1]))
            for i in 0:(order - 1)
                if (r + i * order) % cd_log[2] == 0
                    return abs(div(r + i * order, cd_log[2]))
                end
            end
        end
    end
end
```

Рис. 3.2: Код алгоритма ρ-Метода Полларда для дискретного логарифмирования на Julia



```julia
# Исходные данные a^b = x (mod p)
a1 = 10 # Основание логарифма
b1 = 64 # Число, для которого ищем логарифм
p1 = 107 # Модуль конечного поля
println("Начальные данные: a = $a1, b = $b1, p = $p1, r = $(Find_Order(a1, p1))")

# Вызов функции p-метода Полларда для дискретного логарифмирования
x1 = Pollard_Rho_Method(p1, a1, b1)

println("Дискретный логарифм 10^64 = x (mod 107) равняется: x = $x1")

# Проверка результата
if Power_Mod(a1, x1, p1) == b1
    println("Дискретный логарифм: x = $x1")
else
    println("Решение не найдено")
end
println("\n")


# Исходные данные a^b = x (mod p)
a2 = 2 # Основание логарифма
b2 = 22 # Число, для которого ищем логарифм
p2 = 29 # Модуль конечного поля
println("Начальные данные: a = $a2, b = $b2, p = $p2, r = $(Find_Order(a2, p2))")

# Вызов функции p-метода Полларда для дискретного логарифмирования
x2 = Pollard_Rho_Method(p2, a2, b2)

println("Дискретный логарифм 2^22 = x (mod 29) равняется: x = $x2")

# Проверка результата
if Power_Mod(a2, x2, p2) == b2
    println("Дискретный логарифм: x = $x2")
else
    println("Решение не найдено")
end
```

Рис. 3.3: Начальные данные для нахождения дискретного логарифма на Julia

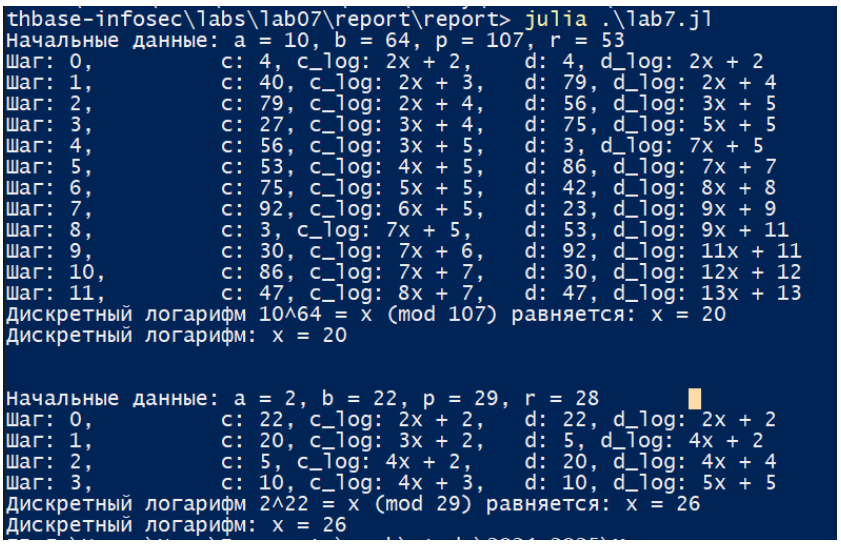


Рис. 3.4: Результат выполнения кода по решению задачи дискретного логарифмирования на Julia

4 Выводы

В ходе выполнения лабораторной работы я изучил работу алгоритмов дискретного логарифмирования в конечном поле — ρ-метода Полларда для дискретного логарифмирования, а также реализовал его программно.

Список литературы

1. Лабораторная работа № 7. Дискретное логарифмирование в конечном поле [Электронный ресурс]. Саратовский государственный университет имени Н.Г. Чернышевского, 2024.