Шаблон отчёта по лабораторной работе №7

Дискретное логарифмирование в конечном поле

Коне Сирики

Содержание

Список иллюстраций

Список таблиц

1 Цель работы

Целью данной лабораторной работы является краткое ознакомление с задачей дискретного логарифмирования и ρ-методом Полларда для её решения, а также его последующая программная реализация.

2 Задание

Рассмотреть и реализовать на языке программирования Python ρ-метод Полларда для задачи дискретного логарифмирования.

3 Теоретическое введение

3.1 Основные понятия из теории групп и теории чисел

Для начала введём некоторые базовые понятия.

Опр. 1. *Группа* – это непустое множество G с бинарной операцией $\cdot$, обладающей свойством ассоциативности ($a \cdot (b \cdot c) = (a \cdot b) \cdot c$) и относительно которой существует нейтральный ($\exists e \in G : \forall a \in G \ ae = ea = a$) и обратный элемент ($\forall a \in G \ \exists a^{-1} \in G : aa^{-1} = a^{-1}a = e$). Если операция $\cdot$ коммутативна, группа называется *абелевой* [1].

Опр. 2. Если $M \subset G$, то *подгруппа, порождённая* M, $\langle M \rangle$, – это пересечение всех групп, содержащих M. Если существует $g \in G$ такой, что $\langle g \rangle = G$, то группа G – циклическая. Все циклические группы абелевы.

Опр. 3. Пусть $m \in \mathbb{N}, m \geq 2$. Целые a и b называются *сравнимыми по модулю* m, если $m \mid (a - b)$, т.е. m является делителем $(a–b)$. Отношение сравнимости записывается следующим образом: $a \equiv b \pmod{m}$ [2].

Опр. 4. Для любого $a \in \mathbb{Z}$ множество чисел $\bar{a} = \{x \in \mathbb{Z} : x \equiv a \pmod{m}\}$ называется *классом вычетов по модулю* m. Существует ровно m классов вычетов по модулю m, причём $\mathbb{Z} = \bar{0} \cup \bar{1} \cup \ldots \cup \overline{(m-1)}$.

Опр. 5. *Кольцо* – это множество с двумя операциями $(R, +, \cdot)$, для которых выполняются свойства: $(R, +)$ – абелева группа, $a \cdot (b \cdot c) = (a \cdot b) \cdot c$, $a \cdot (b + c) =$

$a \cdot b + a \cdot c$ и $(a + b) \cdot c = a \cdot c + b \cdot c$. Кольцо коммутативно, если $\forall a, b \in R$ $a \cdot b = b \cdot a$. Кольцо – с единицей, если $\exists 1 \in R : \forall a \in R : 1 \cdot a = a \cdot 1 = a$ [1].

Опр. 6. *Полем* называется коммутативное кольцо, содержащее не менее двух элементов, в котором все ненулевые элементы образуют группу по умножению [3]. Конечное поле с p элементами, где p – простое число, обозначается $\mathbb{F}_p$ [4].

Опр. 7. Через $\mathbb{Z}_m$ (или $\mathbb{Z}/m\mathbb{Z}$) обозначим *множество классов вычетов по модулю* m: $\mathbb{Z}_m = \{\bar{0}, \bar{1}, \ldots, \overline{m-1}\}$. На нём можно определить операции сложения и умножения: $\bar{a} + \bar{b} = \overline{a + b}$, $\bar{a} \cdot \bar{b} = \overline{a \cdot b}$. $\mathbb{Z}_m$ является коммутативным кольцом с единицей $\bar{1}$, в котором нулевой элемент: $\bar{0}$, а обратный по сложению элемент: $-(\bar{a}) = \overline{(-a)}$. Если m простое, то $\mathbb{Z}_m$ – поле.

Опр. 8. Пусть $\bar{a} \in \mathbb{Z}_m$. Тогда $\bar{b}$ – обратный к $\bar{a}$, если $\bar{a} \cdot \bar{b} = 1$, а $\bar{a}$ является обратимым, если имеет обратный класс. Множество всех обратимых классов в $\mathbb{Z}_m$ обозначается $\mathbb{Z}_m^*$, является группой относительно умножения классов и называется *мультипликативной группой кольца вычетов* $\mathbb{Z}_m$ [5].

3.2 Дискретное логарифмирование

Задача дискретного логарифмирования – наравне с задачей факторизации – является одной из фундаментальных в криптоанализе. На её сложности зиждется стойкость ряда криптосистем, включая такие известные, как:

- схема распределения ключей Диффи-Хеллмана (1976);
- схема Эль-Гамаля (1985), лежащая в основе алгоритма DSA;
- криптосистема Мэсси-Омуры (1978) для передачи сообщений [4].

Для конечного поля $\mathbb{F}_p$ (в частности, в простейшем и важнейшем случае $\mathbb{Z}_p^*$, где p – большое простое число) *задача дискретного логарифмирования* определяется следующим образом [4]: при заданных ненулевых $a, b \in \mathbb{F}_p$ найти такое целое x, что:

$$a^x \equiv b \in \mathbb{F}_p, \text{или } a^x \equiv b \pmod p.$$

Пусть число a также имеет порядок r, то есть $a^r \equiv 1 \pmod p$.

3.3 ρ-метод Полларда для задачи дискретного логарифмирования

Рассмотрим ρ-метод Полларда, который можно применить и для задач дискретного логарифмирования [6]. Здесь, как и в аналогичном методе факторизации, рассмотренном в предыдущей лабораторной, строится последовательность итеративных значений функции f, в которой требуется найти цикл. Для этого, как и ранее, используем алгоритм "черепахи и зайца" Флойда: к одному значению, c, на каждом шаге будем применять функции единожды, к другому, d, – дважды, пока их значения не совпадут и мы не сможем их приравнять.

Так, пусть $c = d \equiv a^{u_0} b^{v_0} \pmod p$, где u_0, v_0 случайные целые числа, – их начальные значения. Поскольку по условию задачи $b \equiv a^x \pmod p$, мы также можем записать $c \equiv a^{u_0}(a^x)^{v_0} \pmod p \equiv a^{u_0+v_0x} \pmod p$. Тогда $\log_a c \pmod p = u_0 + v_0x$. Таким образом, логарифмы c и d по основанию a могут быть представлены линейно.

Теперь зададим отображение f. Оно должно обладать не только сжимающими свойствами, но и вычислимостью логарифма, чтобы по мере изменения значений c и d мы могли также отслеживать изменения в линейном представлении их логарифмов. Будем использовать ветвящееся отображение следующего вида:

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

Таким образом, c будет умножаться или на a, или на b. В первом случае получим $f(c) \equiv (a^u b^v)a \pmod p \equiv a^{u+1}b^v \pmod p \equiv a^{(u+1)+vx} \pmod p$, и тогда $\log_a f(c) = (u+1) + vx = \log_a c + 1$. Во втором случае же получаем $f(c) \equiv (a^u b^v)b \pmod p \equiv a^u b^{v+1} \pmod p \equiv a^{u+(v+1)x} \pmod p$, и отсюда $\log_a f(c) = u+(v+1)x = \log_a c + x$.

Когда значения c и d совпадут, мы сможем приравнять их логарифмы и получим сравнение по x: $u_i^c + v_i^c x \equiv u_i^d + v_i^d x \pmod r$.

Алгоритм 1. 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 $a^x \equiv b \pmod p$, если такой показатель существует.

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

Пример 1. Решим задачу $10^x \equiv 64 \pmod{107}$. Выберем отображение: $f(c) = 10c \pmod{107}$\$ при $c < 53$, $f(c) = 64c \pmod{107}$ при $c \geq 53$. Порядок числа 10 по модулю 107 равен 53. Пусть $u = 2$, $v = 2$. Результаты вычислений представлены в Таблице 3.1.

Таблица 3.1: Применение ρ-метода Полларда для решения примера №1

Шаг	c	$\log_a c$	d	$\log_a d$
0	4	$2+2x$	4	$2+2x$
1	40	$3+2x$	79	$4+2x$
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	$13+13x$

Приравниваем полученные логарифмы: $7+8x \equiv 13+13x \pmod{53}$. Отсюда $-5x \equiv 6 \pmod{53}$. Чтобы решить данное сравнение, нужно найти обратный элемент k^{-1} для $k = -5$ по модулю $m = 53$ ($k^{-1} \cdot k \equiv 1 \pmod{m}$) и умножить на него левую и правую часть сравнения. Так как этот обратный элемент – сравним сам с собой по модулю 53, подобное сравнение будет справедливо [7].

В общем виде пусть решается сравнение $kx \equiv b \pmod{m}$. Если k и m – взаимно простые, т.е. НОД$(k, m) = 1$, мы можем применить расширенный алгоритм Евклида, разобранный в рамках 4-ой лабораторной работы, и получить линейное представление единицы в виде: $s_k \cdot k + s_m \cdot m = 1$ [8]. Отсюда $s_k \cdot k - 1 = -s_m \cdot m$, что эквивалентно $m|(s_k \cdot k - 1)$, что эквивалентно $s_k \cdot k \equiv 1 \pmod{m}$, т.е. $k^{-1} = s_k$. Если же НОД не равен единице, то мы предполагаем, что $gcd =$ НОД$(k, m) =$ НОД(k, b, m) (поскольку в противном случае обратного элемента не существует), и тогда сравнение можно поделить на gcd [7], и получим $\frac{k}{gcd}x \equiv \frac{b}{gcd} \pmod{\frac{m}{gcd}}$.

Возвращаясь к примеру, получаем $x = 20 \pmod{53}$. Проверка: $10^{20} \equiv 64 \pmod{107}$.

4 Выполнение лабораторной работы

Реализуем описанный выше алгоритм на языке **Python** в среде Jupyter Notebook. Для работы нам понадобится функция вычисления порядка числа по модулю, расширенный алгоритм Евклида, реализацию которого мы возьмем из 4-ой лабораторной работы, а также основанная на нём функция решения сравнения вида $k_1 x + b_1 \equiv k_2 x + b_2 \pmod p$:

```
import math
import numpy as np

def multiplicative_order(a, n):
    """
    Вычисляет порядок числа a по модулю n
    """
    k = 1; flag = True # начнем перебор с единицы

    while flag:
        if (a ** k - 1) % n == 0: # если порядок найден
            flag = False # "опускаем" флаг и выходим из цикла
        else: # иначе
            k += 1 # увеличиваем порядок на единицу

    return k
```

```python
def euclidean_algorithm_extended(a, b):
    """
    Находит d = НОД(a, b), а также такие целые числа x и y, что ax + by = d,
    с помощью расширенного алгоритма Евклида
    """
    (a, b) = (int(a), int(b))

    reversed = True if abs(b) > abs(a) else False # флаг
    (a, b) = (b, a) if reversed else (a, b) # меняем местами a и b, если нужно

    (r, x, y) = ([a, b], [1, 0], [0, 1]) # шаг 1

    while r[1] != 0:
        (r[0], r[1], q) = (r[1], r[0] % r[1], r[0] // r[1])

        if r[1] != 0: # если остаток ещё не нулевой..
            (x[0], x[1]) = (x[1], x[0] - q * x[1])
            (y[0], y[1]) = (y[1], y[0] - q * y[1])

    (d, x_r, y_r) = (r[0], x[1], y[1])

    if reversed: # если a и b были в неправильном порядке
        (x_r, y_r) = (y_r, x_r) # меняем найденные коэффициенты местами

    return (d, x_r, y_r)

def solve_congruence(c, d, p):
    """
    Решает сравнение вида k_1 * x + b_1 = k_2 * x + b_2 (mod p), где
```

```
    c = (k_1, b_1), d = (k_2, b_2)
    """
    (k_1, b_1) = c; (k_2, b_2) = d # получаем коэффициенты

    k = k_1 - k_2; b = b_2 - b_1 # kx = b (mod p)
    # k * k_inverse = gcd (mod p)
    (gcd, k_inverse, _) = euclidean_algorithm_extended(k, p)

    if gcd == 1: # если k и p - взаимно простые..
        return (b * k_inverse) % p
    else: # иначе
        k = int(k / gcd); b = int(b / gcd) # делим сравнение на gcd
        (_, k_inverse, _) = euclidean_algorithm_extended(k, int(p / gcd))

        return (b * k_inverse) % p
```

Примеры работы функции `multiplicative_order(a, n)` представлены на Рис. 4.1.

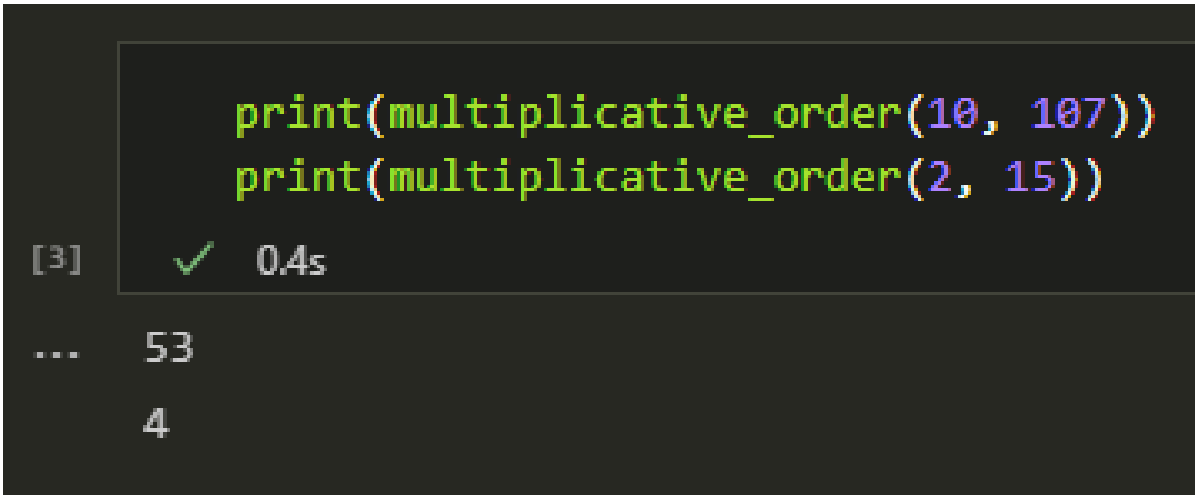

Рис. 4.1: Примеры нахождения порядка числа a по модулю n

4.1 Алгоритм, реализующий ρ-метод Полларда для задачи дискретного логарифмирования

Создадим функцию pollard_rho_method(n, f, c) следующего вида:

```
def pollard_rho_dlog(a, b, p, def0 = True, to_print = False):
    """
    Решает сравнение a^x = b (mod p) ро-методом Полларда;
    def0 = True, если нужно использовать начальные значения u и v по умолчанию,
    и False, если их нужно определить случайно;
    to_print = True, если нужно вывести на экран ход алгоритма
    """
    r = multiplicative_order(a, p) # порядок числа a
    half_p = math.floor(p / 2) # p / 2

    # отображение f
    f = "({a} * x % {p}) if x < {half} else ({b} * x % {p})".format(a = a,
                                              p = p, half = half_p, b = b)

    # начальные значения u и v
    (u, v) = (2, 2) if def0 else (np.random.randint(1, half_p),
                                        np.random.randint(1, half_p))

    if not def0 and to_print:
        print("(u, v) = ({}, {})".format(u, v))

    c = ((a ** u) * (b ** v)) % p #
    d = c                         #
                                  # шаг 1
    (k_c, l_c) = (u, v)           #
```

```
(k_d, l_d) = (u, v)           #

if to_print:
    print("{:^10} | {:^10} | {:^10} | {:^10}"
                                .format("c", "log_c", "d", "log_d"))
    print("{:^10} | {:^10} | {:^10} | {:^10}"
           .format("----------", "----------", "----------", "----------"))
    print("{:^10} | {:^3} + {:^3}x | {:^10} | {:^3} + {:^3}x"
                                .format(c, l_c, k_c, d, l_d, k_d))

while True:
    # вычисляем f(c)
    # и log_a f(c)
    x = c                 #
    if x < half_p:        #
        l_c += 1          #
    else:                 #
        k_c += 1          #
    c = eval(f)           #
                          #
    # вычисляем f(c)      #
    # и log_a f(c)        #
    x = d                 #  шаг 2
    if x < half_p:        #
        l_d += 1          #
    else:                 #
        k_d += 1          #
    x = eval(f)           #
    if x < half_p:        #
```

```
        l_d += 1        #
    else:               #
        k_d += 1        #
    d = eval(f)         #

    if to_print:
        print("{:^10} | {:^3} + {:^3}x | {:^10} | {:^3} + {:^3}x"
                                .format(c, l_c, k_c, d, l_d, k_d))

    # шаг 3
    if c == d:
        # приравниваем логарифмы и решаем сравнение
        result = solve_congruence((k_c, l_c), (k_d, l_d), r)

        if (a ** result - b) % p == 0: # проверка
            return result
        else:
            return 0
```

Теперь с помощью данной функции решим несколько задач на вычисление дискретных логарифмов: $10^x \equiv 64 \pmod{107}$ (см. Рис. 4.2), $5^x \equiv 3 \pmod{23}$ (см. Рис. 4.3) и $29^x \equiv 479 \pmod{797}$ (см. Рис. 4.4).

```
[6]  pollard_rho_dlog(10, 64, 107, True, True)
     ✓ 0.6s

...      c      |   log_c    |     d      |   log_d
    ---------- | ---------- | ---------- | ----------
        4      |  2  +  2 x |     4      |  2  +  2 x
        40     |  3  +  2 x |     79     |  4  +  2 x
        79     |  4  +  2 x |     56     |  5  +  3 x
        27     |  4  +  3 x |     75     |  5  +  5 x
        56     |  5  +  3 x |     3      |  5  +  7 x
        53     |  5  +  4 x |     86     |  7  +  7 x
        75     |  5  +  5 x |     42     |  8  +  8 x
        92     |  5  +  6 x |     23     |  9  +  9 x
        3      |  5  +  7 x |     53     | 11  +  9 x
        30     |  6  +  7 x |     92     | 11  + 11 x
        86     |  7  +  7 x |     30     | 12  + 12 x
        47     |  7  +  8 x |     47     | 13  + 13 x

    20
```

Рис. 4.2: Решение сравнения №1

```
[10]  pollard_rho_dlog(5, 3, 23, False, True)
      ✓ 0.3s

...  (u, v) = (7, 3)
         c      |   log_c    |     d      |   log_d
    ---------- | ---------- | ---------- | ----------
        22     |  3  +  7 x |     22     |  3  +  7 x
        20     |  3  +  8 x |     14     |  3  +  9 x
        14     |  3  +  9 x |     11     |  3  + 11 x
        19     |  3  + 10 x |     4      |  4  + 12 x
        11     |  3  + 11 x |     14     |  5  + 13 x
        10     |  3  + 12 x |     11     |  5  + 15 x
        4      |  4  + 12 x |     4      |  6  + 16 x

    16
```

Рис. 4.3: Решение сравнения №2

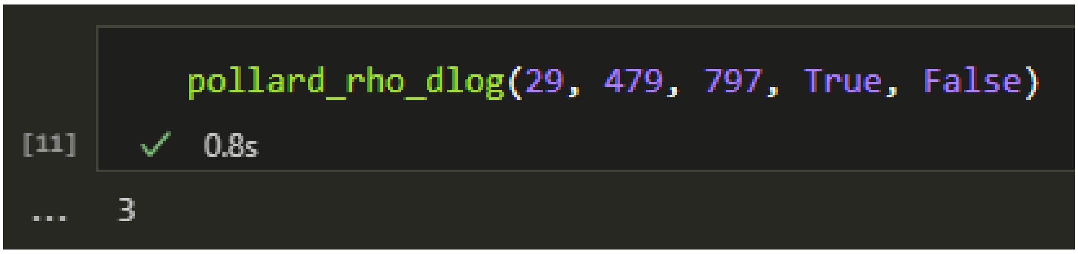

Рис. 4.4: Решение сравнения №3

5 Выводы

Таким образом, была достигнута цель, поставленная в начале лабораторной работы: было проведено краткое знакомство с задачей дискретного логарифмирования и с алгоритмом, реализующим ρ-метод Полларда для её решения, после чего алгоритм был успешно реализован на языке программирования **Python**.

Список литературы

1. Богданов И.И. Теория групп: конспект. ФИВТ МФТИ, 2016. С. 42.
2. Илларионов А.А. Теория чисел: учебное пособие. 2016.
3. Зельвенский И.Г. Группы, кольца, поля: Методические указания по дисциплине «Геометрия и алгебра». Спб.: ГЭТУ ЛЭТИ, 1997. С. 30.
4. Yan S. Primality Testing and Integer Factorization in Public-Key Cryptography. Boston: Springer, 2009. С. 371.
5. Веретенников Б.М., Михалева М.М. Алгебра и теория чисел : учебное пособие. Часть 1 / под ред. Чуксина Н.В. Екатеринбург: Изд-во Урал. ун-та, 2014. С. 52.
6. Бубнов С.А. Лабораторный практикум по основам криптографии: учебно-методическое пособие. Саратов; http://elibrary.sgu.ru/uch_lit/656.pdf: Саратовский государственный университет им. Н.Г. Чернышевского, 2012.
7. Википедия. Сравнение по модулю — Википедия, свободная энциклопедия. 2021.
8. Occhipinti T. Discrete logs with Pollard rho | Math 361. 2021.