Отчет по лабораторной работе №7

Дискретное логарифмирование в конечном поле

Бурдина Ксения Павловна

9 декабря 2023

Содержание

List of Figures

1 Цель работы

Целью данной работы является освоение дискретного логарифмирования в конечном поле, которое применяется во многих алгоритмах криптографии с открытым ключом.

2 Задание

1. Изучить алгоритм дискретного логарифмирования в конечном поле.
2. Реализовать представленный алгоритм и вычислить логарифм по заданным числам p, a, b.

3 Теоретическое введение

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 "$\cdot$". Множество Ω называется *кольцом*, если вполняются следующие условия:

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

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

  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 едининым элемент e такой, что $a \cdot e = e \cdot a = a$ для любого $a \in \Omega$, то кольцо называется кольцом с единицей.

Если в ассоциативном, коммутативном кольце Ω с единицей 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 $m \in N, m > 1$. 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:

- *Рефлексивность*: $a \equiv a\ (mod\ m)$.
- *Симметричность*: если $a \equiv b\ (mod\ m)$, то $b \equiv a\ (mod\ m)$.
- *Транзитивность*: если $a \equiv b\ (mod\ m)$ и $b \equiv c\ (mod\ m)$, то $a \equiv c\ (mod\ m)$.

Отношение, обладающее свойством рефлесивности, симметриности и транзитивности, называется *отношением эквивалентности*. Отношение сравнимости является отношением эквивалентности на множестве Z целых чисел [2].

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, обознаается $a\ (mod\ m)$ или $\bar{a}$ и представляет собой множество чисел вида $a + km$, где $k \in Z$; число a называется представителем этого класса вычетов.

Множество классов вычетов по модулю m обозначается Z/mZ, состоит ровно из m элементов и относительно операций сложения и умножения является *кольцом классов вычетов* по модулю m.

Пример. Если $m = 2$, то $Z/2Z = \{0(mod2), 1(mod2)\}$, где $0(mod2) = 2Z$ - множество всех четных чисел, $1(mod2) = 2Z + 1$ - множество всех нечетных чисел.

Обозначим $F_p = Z/pZ$, 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 $x = log_a b$.

Безопастность соответствующих криптосистем основана на том, что, зная числа a, x, p вычислить $a^x\ (mod\ p)$ легко, а решить задачу дискретного логарифмирования трудно. Рассмотрим p-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:

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

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

3.1 Алгоритм, реализующий p-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 $a^x \equiv b\ (mod\ p)$, если такой показатель существует.

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

Пример [1]. Решим задачу дискретного логарифмирования $10^x \equiv 64 \ (mod\ 107)$, используя p-Метод Полларда. Порядок числа 10 по модулю 107 равен 53.

Выберем отображение $f(c) = 10c \ (mod\ 107)$ при $c < 53$, $f(c) = 64c \ (mod\ 107)$ 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

Figure 3.1: Схема работы алгоритма

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

Проверка: $10^{20} \equiv 64 \ (mod\ 107)$.

4 Ход выполнения лабораторной работы

Для реализации рассмотренного алгоритма разложения чисел на множители будем использовать среду JupyterLab. Выполним необходимую задачу.

1. Пропишем алгоритм Евклида, который был показан в предыдущих лабораторных работах, а также запишем функцию для вывода его инверсивного значения:

```
def alg_e_ext(a, b):
    if b == 0:
        return a, 1, 0
    else:
        d, x, y = alg_e_ext(b, a % b)
    return d, y, x - (a // b) * y
```

```
def inv(a, n):
    return alg_e_ext(a, n)[1]
```

Figure 4.1: Расширенный алгоритм Евклида

2. Также пропишем функцию для подсчета значений при выполнении алгоритма Полларда:

```
def fun(x, a, b, xxx):
    (G, H, P, Q) = xxx
    sub = x % 3
    if sub == 0:
        x = x * xxx[0] % xxx[2]
        a = (a + 1) % Q
    if sub == 1:
        x = x * xxx[1] % xxx[2]
        b = (b + 1) % xxx[2]
    if sub == 2:
        x = x * x % xxx[2]
        a = a * 2 % xxx[3]
        b = b * 2 % xxx[3]
    return x, a, b
```

Figure 4.2: Вспомогательная функция

3. Запишем алгоритм, реализующий *p-метод Полларда*, с помощью следующей функции:

```
def Pollard(G, H, P):
    Q = int((P - 1) // 2)
    x = G * H
    a = 1
    b = 1
    X = x
    A = a
    B = b
    for i in range(1, P):
        x, a, b = fun(x, a, b, (G, H, P, Q))
        X, A, B = fun(X, A, B, (G, H, P, Q))
        X, A, B = fun(X, A, B, (G, H, P, Q))
        if x == X:
            break
    nom = a - A
    denom = B - b
    res = (inv(denom, Q) * nom) % Q
    if prov(G, H, P, res):
        return res
    return res + Q
```

Figure 4.3: p-Метод Полларда

Здесь на вход подается 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. Необходимо выполнить следующее:

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

По итогу при вызове функции мы получим показатель x, для которого $a^x \equiv b \ (mod\ p)$, если такой показатель существует.

4. Проверим корректность работы алгоритма для заданных сведений. Для этого запишем условие примера с помощью следующей функции:

```
def prov(g, h, p, x):
    return pow(g, x, p) == h
args = [(10, 64, 107)]
for arg in args:
    res = Pollard(*arg)
    print(arg, ' : ', res)
    print("Validates: ", prov(arg[0], arg[1], arg[2], res))
```

```
(10, 64, 107)  :  20
Validates:  True
```

Figure 4.4: Пример работы алгоритма

При вызове данной функции видим, что получаем то же число, что было описано в примере. То есть $x = 20 \ (mod\ 53)$ для задачи дискретного логарифмирования $10^x \equiv 64 \ (mod\ 107)$.

5 Листинг программы

```
def alg_e_ext(a, b):
  if b == 0:
    return a, 1, 0
  else:
    d, x, y = alg_e_ext(b, a % b)
  return d, y, x - (a // b) * y

def inv(a, n):
  return alg_e_ext(a, n)[1]

def fun(x, a, b, xxx):
  (G, H, P, Q) = xxx
  sub = x % 3
  if sub == 0:
    x = x * xxx[0] % xxx[2]
    a = (a + 1) % Q
  if sub == 1:
    x = x * xxx[1] % xxx[2]
    b = (b + 1) % xxx[2]
  if sub == 2:
    x = x * x % xxx[2]
    a = a * 2 % xxx[3]
```

```
    b = b * 2 % xxx[3]
  return x, a, b

def Pollard(G, H, P):
  Q = int((P - 1) // 2)
  x = G * H
  a = 1
  b = 1
  X = x
  A = a
  B = b
  for i in range(1, P):
    x, a, b = fun(x, a, b, (G, H, P, Q))
    X, A, B = fun(X, A, B, (G, H, P, Q))
    X, A, B = fun(X, A, B, (G, H, P, Q))
    if x == X:
      break
  nom = a - A
  denom = B - b
  res = (inv(denom, Q) * nom) % Q
  if prov(G, H, P, res):
    return res
  return res + Q

def prov(g, h, p, x):
  return pow(g, x, p) == h
args = [(10, 64, 107)]
for arg in args:
  res = Pollard(*arg)
```

```
print(arg, ' : ', res)
print("Validates: ", prov(arg[0], arg[1], arg[2], res))
```

6 Выводы

В ходе работы мы изучили и реализовали дискретное логарифмирование в конечном поле.

7 Список литературы

1. Фороузан Б. А. Криптография и безопасность сетей. - М.: Интернет-Университет Информационных Технологий : БИНОМ. Лаборатория знаний, 2010. - 784 с. [1]

2. Методические материалы курса [2]