Лабораторная работа 7

Дискретное логарифмирование в конечном поле

Греков Максим Сергеевич

Содержание

List of Figures

1 Цель работы

- Ознакомиться с задачей дискретного логарифмирования в конечном поле
- Рассмотреть теоретические основы представленного алгоритма
- Реализовать ρ-Метод Полларда для задач дискретного логарифмирования

2 Описание задачи

2.1 Введение

Задача дискретного логарифмирования, как и задача разложения на множители, применяется во многих алгоритмах криптографии с открытым ключом.

Предложена 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.

2.2 Условия кольца

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:

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

для любых $a, b, c \in \Omega$.

2.3 Свойства кольца

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.

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*.

2.4 Свойства отношения сравнимости

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:

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

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

2.5 Классы эквивалентности

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 $\bar{a}$ 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.

2.6 Постановка задачи

Пример. Если $m = 2$, то Z/2Z= $\{0\ (mod\ 2), 1\ (mod\ 2)\}$, где $0\ (mod\ 2) = 2Z$ – множество всех четных чисел, $1\ (mod\ 2) = 2Z + 1$ – 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 $x = \log_a b$.

3 ρ-Метод Полларда

3.1 Отображение f()

Безопасность соответствующих криптосистем основана на том, что, зная числа a, x, p вычислить $a^x \, (mod \, p)$ легко, а решит 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:

3.2 Описание алгоритма

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

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

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.

3.3 Последовательность вычислений

1. Выбрать произвольные целые числа u, v и положить $c \leftarrow a^u b^v\ (mod\ p), d \leftarrow c$.
2. Выполнять $c \leftarrow f(c)(mod\ p), d \leftarrow f\big(f(d)\big)(mod\ p)$, 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".

3.4 Пример

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

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

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

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

4 Реализация алгоритма

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


def func(x, a, b, G, H, P, Q):
    sub = x % 3

    if sub == 0:
        x = x * G % P
        a = (a + 1) % Q

    if sub == 1:
        x = x * H % P
        b = (b + 1) % P

    if sub == 2:
        x *= x % P
        a *= 2 % Q
        b *= 2 % Q

    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 = func(x, a, b, G, H, P, Q)
        X, A, B = func(X, A, B, G, H, P, Q)
        X, A, B = func(X, A, B, G, H, P, Q)

        if x == X: break

    nom = a-A
    denom = B-b

    return (euclid(denom, Q)[1] * nom) % Q


def main():
    a = 10
    b = 64
    p = 107

    x = pollard(a, b, p)
    print(f"{a}^{x} = {b}(mod {p})")


if __name__ == '__main__':
    main()
```

Figure 4.1: Код на языке Python

4.1 Результат работы

Результат работы данного алгоритма:

$$10^{20} = 64(mod 107)$$

4.2 Программный код

```
def euclid(a, b):
    if b == 0:
        return a, 1, 0
```

```
    else:
        d, x, y = euclid(b, a % b)
        return d, y, x - (a // b) * y


def func(x, a, b, G, H, P, Q):
    sub = x % 3

    if sub == 0:
        x = x * G % P
        a = (a + 1) % Q

    if sub == 1:
        x = x * H % P
        b = (b + 1) % P

    if sub == 2:
        x *= x % P
        a *= 2 % Q
        b *= 2 % Q

    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 = func(x, a, b, G, H, P, Q)
        X, A, B = func(X, A, B, G, H, P, Q)
        X, A, B = func(X, A, B, G, H, P, Q)

        if x == X: break

    nom = a-A
    denom = B-b

    return (euclid(denom, Q)[1] * nom) % Q


def main():
    a = 10
    b = 64
    p = 107

    x = pollard(a, b, p)
    print(f"{a}^{x} = {b}(mod {p})")


if __name__ == '__main__':
```

main()

5 Выводы

- Ознакомились с задачей дискретного логарифмирования в конечном поле
- Рассмотрели теоретические основы представленного алгоритма
- Реализовали p-Метод Полларда для задач дискретного логарифмирования