Отчёт по лабораторной работе №7

Дискретное логарифмирование

Бакундукизе Эжид Принц НФИмд-01-21

Содержание

List of Figures

1 Цель работы

Изучение задачи дискретного логарифмирования.

2 Теоретические сведения

Пусть в некоторой конечной мультипликативной абелевой группе G задано уравнение

$$g^x = a$$

Решение задачи дискретного логарифмирования состоит в нахождении некоторого целого неотрицательного числа x, удовлетворяющего уравнению. Если оно разрешимо, у него должно быть хотя бы одно натуральное решение, не превышающее порядок группы [1]. Это сразу даёт грубую оценку сложности алгоритма поиска решений сверху — алгоритм полного перебора нашёл бы решение за число шагов не выше порядка данной группы.

Чаще всего рассматривается случай, когда группа является циклической, порождённой элементом g. В этом случае уравнение всегда имеет решение. В случае же произвольной группы вопрос о разрешимости задачи дискретного логарифмирования, то есть вопрос о существовании решений уравнения , требует отдельного рассмотрения.

2.1 p-алгоритм Полларда

- Вход. Простое число p, число a порядка r по модулю p, целое число bб $1 < b < p$; 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 $c = a^u b^v (mod p), d = c$
2. Выполнять $c=f(c)(mod p), d=f(f(d))(mod p), вычисляя при этом логарифмы для c и d как линейные функции от x по модулю r, до получения равенства $c = d(mod p)$
3. Приняв логарифмы для c и d, вычислить логарифм x решением сравнения по модулю r. Результат x или РЕШЕНИЯ НЕТ.

3 Выполнение работы

3.1 Реализация алгоритма на языке Python

```
# p - простое число (107)
# a - число порядка r по модулю p (число a=10 порядка  r = 53 по модулю p=107)
# b - целое число от 1 до p = 107
# f - отображение, обаладающее сжимающими св-вами и сохраняющее вычислимость лога


def f(p, a, b, x, u, v):
    if x < 53:
        return pow(a*x,1,p), u+1, v
    else:
        return pow(b*x,1,p), u, v+1


def ext_Euc(a, b):
    rp = a
    rc = b
    xp, xc = 1, 0
    yp, yc = 0, 1
    rn = rp % rc
    d = rc
```

```
    while rn != 0:
        rn = rp % rc
        q = (rp - rn)/rc
        d, x, y = rc, xc, yc

        rp = rc
        rc = rn

        xc = xp - q*xc
        xp = x

        yc = yp - q*yc
        yp = y

    return d, x, y


def Pol(p, a, r, b, u, v):
    c = pow(a**u * b**v,1,p)
    d = c
    uc, vc = u, v
    ud, vd = u, v

    c, uc, vc = f(p, a, b, c, uc, vc)
    c %= p
    d, ud, vd = f(p, a, b, *f(p, a, b, d, ud, vd))
    d %= p

    while c%p != d%p:
```

```
        c, uc, vc = f(p, a, b, c, uc, vc)
        c %= p
        d, ud, vd = f(p, a, b, *f(p, a, b, d, ud, vd))
        d %= p

    v = vc - vd
    u = ud - uc

    d, x, y = ext_Euc(v, r)

    while d != 1:
        v /= d
        u /= d
        r /= d
        d, x, y = ext_Euc(v, r)

    return x*u % r
```

3.2 Контрольный пример

```
In [70]: p = 107
         a = 10
         b = 64
         r = 53
         u = 2
         v = 2


         print('Результат ', Pol(p, a, r, b, u, v))

         Результат  20.0
```

Figure 3.1: Работа алгоритма

4 Выводы

В ходе выполнения данной лабораторной работы мы познакомились с дискретным логарифмированием в конечном поле, программно реализовали алгоритм Полларда.

Список литературы

1. Дискретное логарифмирование)
2. Доступно о криптографии на эллиптических кривых