Отчёт по лабораторной работе №7

Дискретное логарифмирование в конечном поле

Милёхин Александр НПМмд-02-21

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. 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 р-алгоритм Полларда

- 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 $a^x = b(modp)$, если такой показатель существует.

1. Выбрать произвольные целые числа u, v и положить $c = a^u b^v (modp), d = c$
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 $c = d(modn)$
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 алгоритмов на языке Python

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


def inverse(a, n):
    return  (euclid_extended(a, n)[1])


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

```
        b = (b + 1) % x_swap[2]
    if sub == 2:
        x = x*x % x_swap[2]
        a = a*2 % x_swap[3]
        b = b*2 % x_swap[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 = xab(x, a, b, (G, H, P, Q))
        X, A, B = xab(X, A, B, (G, H, P, Q))
        X, A, B = xab(X, A, B, (G, H, P, Q))
        if x == X:
            break
    nom = a-A
    denom = B-b
    res = (inverse(denom, Q) * nom) % Q
    if verify(G, H, P, res):
        return res
    return res + Q


def verify(g, h, p, x):
```

```
    return pow(g, x, p) == h

args = [(10, 64, 107)]

for arg in args:
    res = pollard(*arg)
    print(arg, ':     x =', res)
    print("Верификация: ", verify(arg[0], arg[1], arg[2], res))
    print()
```

3.2 Контрольный пример

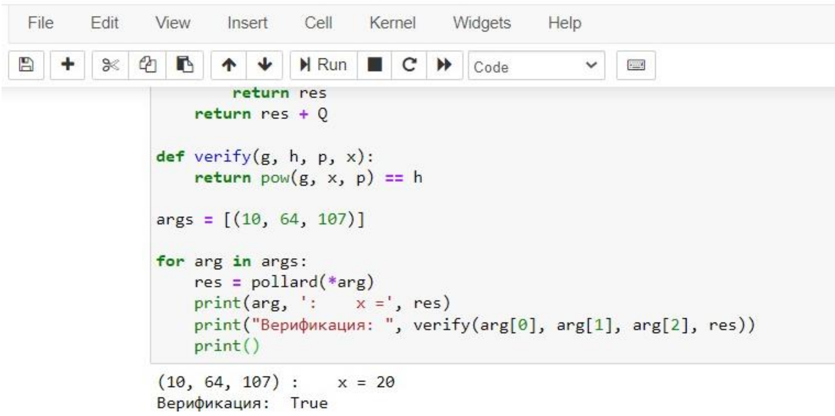

Figure 3.1: Пример работы алгоритма

Получаем x = 20 для значений данном примере.

4 Выводы

Я изучил задачу дискретного логарифмирования, повторил р-алгоритм Полларда, а также реализовал алгоритм программно на языке Python.

Список литературы

1. Дискретное логарифмирование)

2. Доступно о криптографии на эллиптических кривых