

# **Отчёт по лабораторной работе №7**

**Дискретное логарифмирование**

Мохаммад хоссейн фарзанфар

# Содержание

<b>Цель работы</b>	<b>4</b>
<b>Теоретические сведения</b>	<b>5</b>
р-алгоритм Поллрада . . . . .	5
<b>Выполнение работы</b>	<b>7</b>
Реализация алгоритма на языке Python . . . . .	7
Контрольный пример . . . . .	11
<b>Выводы</b>	<b>12</b>
<b>Список литературы</b>	<b>13</b>

# Список иллюстраций

1	Работа алгоритма . . . . .	11
---	----------------------------	----

# Цель работы

Изучение задачи дискретного логарифмирования.

# 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.

## **p-алгоритм Поллрада**

- 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  $a^x = b(mod p)$ , 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.

# Выполнение работы

## Реализация алгоритма на языке Python

```
def ext_euclid(a, b):
    """
    Extended Euclidean Algorithm
    :param a:
    :param b:
    :return:
    """
    if b == 0:
        return a, 1, 0
    else:
        d, xx, yy = ext_euclid(b, a % b)
        x = yy
        y = xx - (a // b) * yy
        return d, x, y

def inverse(a, n):
    """
    Inverse of a in mod n
    :param a:
```

```

:param n:
:return:
"""
return ext_euclid(a, n)[1]

```

```

def xab(x, a, b, xxx_todo_changeme):
    """
    Pollard Step
    :param x:
    :param a:
    :param b:
    :return:
    """
    (G, H, P, Q) = xxx_todo_changeme
    sub = x % 3 # Subsets

    if sub == 0:
        x = x*xxx_todo_changeme[0] % xxx_todo_changeme[2]
        a = (a+1) % Q

    if sub == 1:
        x = x * xxx_todo_changeme[1] % xxx_todo_changeme[2]
        b = (b + 1) % xxx_todo_changeme[2]

    if sub == 2:
        x = x*x % xxx_todo_changeme[2]
        a = a*2 % xxx_todo_changeme[3]
        b = b*2 % xxx_todo_changeme[3]

```



```
return x, a, b
```

```
def pollard(G, H, P):
```

```
    # P: prime
```

```
    # H:
```

```
    # G: generator
```

```
    Q = int((P - 1) // 2) # sub group
```

```
    x = G*H
```

```
    a = 1
```

```
    b = 1
```

```
    X = x
```

```
    A = a
```

```
    B = b
```

```
    # Do not use range() here. It makes the algorithm amazingly slow.
```

```
    for i in range(1, P):
```

```
        # Who needs pass-by reference when you have Python!!! ;)
```

```
        # Hedgehog
```

```
        x, a, b = xab(x, a, b, (G, H, P, Q))
```

```
        # Rabbit
```

```
        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

# print nom, denom

# It is necessary to compute the inverse to properly compute the fraction mod q
res = (inverse(denom, Q) * nom) % Q

# так никто не делает но все же...
if verify(G, H, P, res):
    return res

return res + Q

def verify(g, h, p, x):
    """
    Verifies a given set of g, h, p and x
    :param g: Generator
    :param h:
    :param p: Prime
    :param x: Computed X
    :return:

```

```

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

args = [
    (10, 64, 107),
]

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

```

## Контрольный пример



```

111 args = [
112     (10, 64, 107),
113 ]
114
115 for arg in args:
116     res = pollard(*arg)
117     print(arg, ': ', res)
118     print("Validates: ", verify(arg[0], arg[1], arg[2], res))
119     print()

```

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

Рис. 1: Работа алгоритма

# Выводы

Изучили задачу дискретного логарифмирования.

## Список литературы

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