МИНОБРНАУКИ РОССИИ
ФГБОУ ВО «СГУ ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»

ЛАБОРАТОРНАЯ РАБОТА №2

студента 4 курса 431 группы
направления 10.05.01 — Компьютерная безопасность
факультета КНиИТ
Никитина Арсения Владимировича

Проверил
доцент ____________ А. В. Жаркова

Саратов 2022

СОДЕРЖАНИЕ

1 Задание лабораторной работы

Вычислить методом Сильвера-Полига-Хеллмана дискретный логарифм элемента 25 в поле F_{41} относистельно порождающего элемента $g = 7$.

2 Теоретическая часть

Пусть F_q — конечное поле порядка $q = p^r$. Пусть $q - 1 = \prod_p p^{\alpha_p}$ — разложение числа $q - 1$ в произведение степеней простых чисел.

Предположим, что все простые делители числа $q - 1$ малы. В этом случае примарное число q называется гладким. При такой предположениии существует достаточно быстрый алгоритм нахождения дискретного логарифма в F_q^*.

Рассмотрим возможную атаку на дискретный логарифм для гладких q.

Алгоритм Сильвера-Полига-Хеллмана

Пусть g — порождающий элемент в F_q^*.

В начале для каждого простого делителя p числа $q-1$ вычислим все корни $p-$й степени из 1:

$$r_{p,j} = g^{j(q-1)/p},\ j = \overline{0, p-1}.$$

Поскольку элемент g является порождающим элементом мультипликативной группы F_q^*, все выписанные элементы различны. Легко проверить, что они являются корнями степени p из 1: $r_{p,j}^p = g^{j(q-1)} = 1,\ j = \overline{0, p-1}$ по Малой теореме Ферма. В любом поле решений уравнения степени p не более чем p, а, значит, составленный список полон и больше в нем элементов не прибавится.

Этот шаг проделывается один раз. Его результат — списки значений $r_{p,j},\ j = \overline{0, p-1}$ сохраняются. Обозначим такой список как $\sqrt[p]{1}$.

Рассмотим проблему нахождеия дискретного логарифма:

$$x : g^x = y$$

при различных значениях $y \in F_q^*$.

Заметим, что нам достаточно найти $\forall p \mid (q-1)$ вычет $x(p) = x(mod\ p^{\alpha_p})$. Полученные значения $x(p)$ позволяют записать все сравнения $x = x(p)(mod\ p^{\alpha_p})$ и вычислить x по Китайской теореме об остатках.

Будем использовать $p-$ичную запись числа $x(p)(mod\ p^{\alpha_p})$.

Предположим, что:

$$x(p) = x_0 + x_1 p + x_2 p^2 + ... + x_{\alpha_p - 1} p^{\alpha_p - 1}\ (mod\ p^{\alpha_p}),\ 0 \leqslant x_i < p.$$

Для того, чтобы вычислить x_0, подсчитаем вначале $y^{(q-1)/p}$. В результате мы получим корень $p-$й степени из 1 в F_q^*. Так как $y = g^x$, то $y^{(q-1)/p} = g^{x(q-1)/p} = g^{x_0(q-1)/p} = r_{p,x_0}$.

Сравниваем r_{p,x_0} с элементами $r_{p,j}$, где $0 \leqslant j < p$. Находим такое значение j, что $r_{p,x_0} = r_{p,j}$. Значит, $x_0 = j$.

Для того, чтобы вычислить x_1, заменим y на:

$$y_1 = yg^{-x_0} = g^{x_0+x_1p+...+x_{\alpha-1}p^{\alpha_p-1}}.$$

Новый элемент y_1 имеет дискретный логарифм, равный:

$$x - x_0 = x_1p + ... + x_{\alpha_{p-1}}p^{\alpha_p-1}\ (mod\ p^{\alpha_p}).$$

Ясно, что $y_1^{(q-1)/p} = 1$. Далее:

$$y_1^{(q-1)/p^2} = g^{(x-x_0)(q-1)/p^2} = g^{(x_1+x_2p+...)(q-1)/p} = g^{x_1(q-1)/p} = r_{p,x_1}.$$

Сравниваем r_{p,x_1} с элементами $\{r_{p,j}\}$. Находим $j : r_{p,x_1} = r_{p,j}$. Полагаем, $x_1 = j$. Продолжаем этот процесс, находим значение:

$$x = x(p) = x_0 + x_1p + ... + x_{\alpha_{p-1}}p^{\alpha_{p-1}}\ ((mod)\ p^{\alpha_p})$$

Далее записываем систему сравнений:

$$x \equiv x_p\ (mod\ p^{\alpha_p})$$

для всех простых делителей p числа $q-1$ и находим x, используя Китайсткую теорему об остатках

3 Практическая часть

3.1 Решение задачи

$\underset{a}{7}^{x} \equiv \underset{b}{25} \pmod{\underset{p}{41}}$

$b^{\frac{p-1}{q}} \equiv a^{\frac{p-1}{q} \cdot x_0} \pmod{p}$

$b_i = b_{i-1} \cdot a^{-x_{i-1} \cdot q^{i-1}}$

$p - 1 = 40 = 2^3 \cdot 5^1$

$\left(a^{\frac{p-1}{q}}\right)^{x_i} \equiv b_i^{\frac{p-1}{q^{i+1}}}$

Р-м $q = 2$, $\alpha = 3$

$q_i \backslash i$	0	1	2	3
2	1	40	1	40
5	1	37		

x_0: $7^{20x_0} \equiv 25^{20} \equiv 1 \Rightarrow x_0 = 0$

$b_1 = 25 \cdot 7^{-0 \cdot 2^0} \equiv 25$

x_1: $7^{20x_1} \equiv 25^{40/4} \equiv 25^{10} \equiv 1 \Rightarrow x_1 = 0$

$b_2 = 25 \cdot 7^{-0 \cdot 2^1} = 25$

x_2: $7^{20x_2} \equiv 25^{40/8} \equiv 25^5 \equiv 40 \Rightarrow x_2 = 1$

$\Rightarrow x \equiv 0 \cdot 2^0 + 0 \cdot 2^1 + 1 \cdot 2^2 \pmod{8} \equiv \boxed{4 \bmod 8}$

Р-м $q = 5$, $\alpha = 1$

x_0: $7^{8x_0} \equiv 25^8 \equiv 37 \Rightarrow x_0 = 1$

$\Rightarrow x \equiv x_0 \cdot 5^0 \equiv 1 \cdot 5^0 \equiv \boxed{1 \bmod 5}$

$\Rightarrow \begin{cases} x \equiv 4 \pmod{8} \\ x \equiv 1 \pmod{5} \end{cases} \Rightarrow x \equiv 36 \pmod{41}$

Ответ: $x = 36$

3.2 Пример работы алгоритма

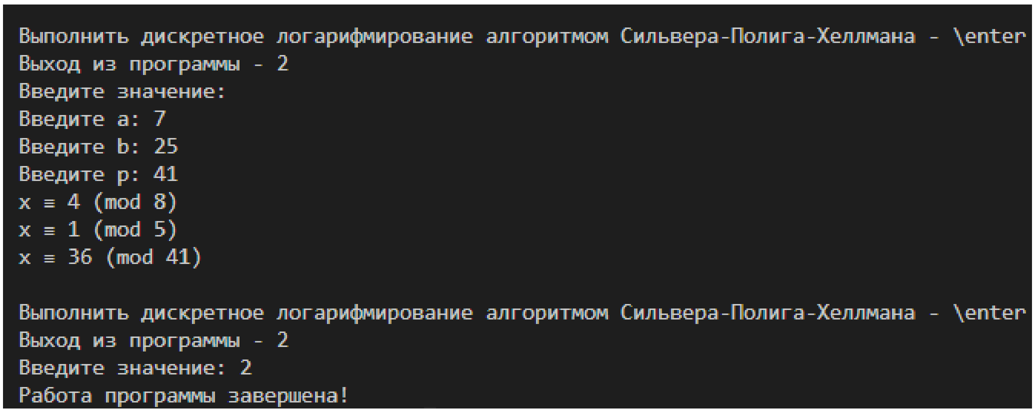

Рисунок 1

3.3 Код программы, реализующей рассмотренный алгоритм

```
from sympy import *

from functools import reduce

def gcd(a, b):
    while b != 0:
        a, b = b, a % b
    return a


def coprime(a, b):
    return gcd(a, b) == 1


def get_parent_elements(m):

    counter = 1
    res = []

    for i in [i for i in range(2, m) if coprime(i, m)]:

        current_counter = 1
        elem_save = elem = i

        while elem != 1:
            elem *= elem_save
            elem %= m
```

```
            current_counter += 1

        if current_counter == counter:
            res.append(i)
        elif current_counter > counter:
            res = [i]
            counter = current_counter

    return res

# number, modula!
def gcdExtended(a, b):
    if a == 0 :
        return b, 0, 1

    gcd, x1, y1 = gcdExtended(b % a, a)

    x = y1 - (b // a) * x1
    y = x1

    return gcd, x, y


def chinese_remainder(pairs):

    mod_list, remainder_list = [p[0] for p in pairs], [p[1] for p in pairs]
    mod_product = reduce(lambda x, y: x * y, mod_list)
    mi_list = [mod_product // x for x in mod_list]
    mi_inverse = [gcdExtended(mi_list[i], mod_list[i])[1] for i in
     ↪  range(len(mi_list))]
    x = 0
    for i in range(len(remainder_list)):
        x += mi_list[i] * mi_inverse[i] * remainder_list[i]
        x %= mod_product
    return x


def factor(p):

    d, factors, unique_factors = 2, [], set()

```

```
    while d*d <= p:

        if isprime(d):
            while (p % d) == 0:
                factors.append(d)
                unique_factors.add(d)
                p //= d

        d += 1


    if p > 1:
       unique_factors.add(p)
       factors.append(p)

    return [(i, factors.count(i)) for i in unique_factors]


def modula_power(a, power, modula):

    b = 1
    while power:
        if not power % 2:
            power //= 2
            a = (a * a) % modula
        else:
            power -= 1
            b = (b * a) % modula
    return b


def powers_table(factors_powers, a, p):

    table = []

    for (factor, _) in factors_powers:

        subres = []

        for i in range(p):

```

```
            probable_value = ((factor, i, modula_power(a, ((i * (p - 1)) //
             ↪ factor ), p)))

            if (k := probable_value[2]) not in subres:
                table.append(probable_value)
                subres.append(k)

    return table


def find_in_table(table, q, value):
    for (a, b, c) in table:
        if a == q and c == value:
            return b


def decomposition(a_inversed, b, q_power, p, table):

    q, power = q_power[0], q_power[1]

    right = modula_power(b, (p - 1) // q, p)

    x_0 = find_in_table(table, q, right)


    if power == 1:

        print(f'x \u2261 {x_0} (mod {q ** power})')
        return x_0

    else:
        congruence = [x_0]

        for i in range(1, power):

            b = (b * modula_power(a_inversed, x_0 * (q ** (i - 1)), p)) % p
            right = modula_power(b, (p - 1) // (q ** (i + 1)), p)

            x_0 = find_in_table(table, q, right)
            congruence.append(x_0 * (q ** i))

```

```
        res = sum(congruence) % (q ** power)
        print(f'x \u2261 {res} (mod {q ** power})')

        return res


def get_number(pos, parent_elems = None):

    while True:
        try:
            if pos == 'a':
                print('Порождающие элементы циклической группы: ', *parent_elems)

            number = int(input(f'Введите {pos}: '))

            if number > 0:

                if pos == 'p':
                    if isprime(number):
                        return number
                    else:
                        print('Вы ввели не простое число')
                elif pos == 'a':
                    # if number in parent_elems:
                        # return number
                    # else:
                        # print('Вы ввели не порождающий элемент')
                        return number
                else:
                    return number

            else:
                print('Вы ввели не положительное целое число')
        except ValueError:
            print('Вы ввели не число')


def main():

    while True:
```

```python

        print('\nВыполнить дискретное логарифмирование алгоритмом '\
            'Сильвера-Полига-Хеллмана - \enter')
        print('Выход из программы - 2')

        try:
            value = int(input('Введите значение: '))
        except ValueError:
            value = 1

        if value == 1:

            b = get_number('b')
            p = get_number('p')
            a = get_number('a', get_parent_elements(p))

            # if a not in get_parent_elements(p):
            if modula_power(a, p - 1, p) != 1:
                print(f'Число {a} не является порождающим элементом циклической группы F_{p}')
                continue

            factors_powers = factor(p - 1)

            table = powers_table(factors_powers, a, p)

            a_inversed = (gcdExtended(a, p)[1] % p + p) % p

            congruences = [(q_power[0] ** q_power[1], \
                decomposition(a_inversed, b, q_power, p, table))  \
                    for q_power in factors_powers]

            print(f'x \u2261 {chinese_remainder(congruences)} (mod {p})')

        if value == 2:
            print('Работа программы завершена!')
            break


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

229