Отчёт по лабораторной работе №7

Ван Яо

Содержание

Список таблиц

Список иллюстраций

Цель работы

1. Изучить теоретические основы задачи дискретного логарифмирования в конечных полях.
2. Реализовать программно алгоритм Полларда (ρ-метод) для решения задачи дискретного логарифмирования.
3. Провести тестирование алгоритма на примерах.

Ход лабораторной работы

Теоретические основы

1. Задача дискретного логарифмирования (DLP)

Пусть p — простое число,a элемент мультипликативной группы конечного поля $F_p^* = \mathbb{Z}/p\mathbb{Z} \setminus \{0\}$,имеющий порядок r.Для заданного $b \in \langle a \rangle$. (циклической подгруппе, порождённой a) требуется найти целое число x такое, что:

$$a^x \equiv b \pmod{x}, \quad 0 \leq x < r$$

Число x называется **дискретным логарифмом** b по основанию a по модулю p и обозначается $x = \log_a b$.

2. ρ -метод Полларда для DLP

Алгоритм основан на парадоксе дней рождения и методе поиска циклов Флойда. Идея — построить псевдослучайную последовательность элементов группы, в которой рано или поздно произойдёт коллизия (повторение значения). При этом каждое значение последовательности представляется в виде $a^{\alpha_i} b^{\beta_i}$,что позволяет при коллизии получить уравнение относительно x.

Алгоритм:

- Разбить группу F_p^* на три непересекающихся подмножества S_1, S_2, S_3 (например, по остатку от деления на 3).

- Определить функцию перехода:

$$f(c) = \begin{cases} c \cdot a \bmod p, & c \in S_1 \\ c \cdot b \bmod p, & c \in S_2 \\ c^2 \bmod p, & c \in S_3 \end{cases}$$

Соответственно обновляются показатели (α, β).

- Инициализировать два указателя (“черепаху” и “зайца”) с начальным состоянием $(c, \alpha, \beta) = (1, 0, 0)$.
- На каждой итерации:
  - Черепаха делает один шаг:$(c_t, \alpha_t, \beta_t) = f(c_t, \alpha_t, \beta_t)$
  - Заяц делает два шага:$(c_h, \alpha_h, \beta_h) = f(f(c_h, \alpha_h, \beta_h))$
- Если $c_t = c_h$,то получаем равенство:

$$a^{\alpha_t} b^{\beta_t} \equiv a^{\alpha_h} b^{\beta_h} \pmod p \Rightarrow a^{\alpha_t - \alpha_h} b^{\beta_t - \beta_h} \pmod p$$

Подставля $b = a^x$,получаем линейное сравнение:

$$(\beta_t - \beta_h)x \equiv (\alpha_h - \alpha_t) \pmod r$$

- Решить это сравнение с помощью расширенного алгоритма Евклида. Проверить найденное решение подстановкой.

Практическая реализация

Вспомогательные функции

Реализация расширенного алгоритма Евклида

```
def egcd(a,b):
    if a==0:
        return b,0,1
    g,x1,y1=egcd(b%a,a)
    return g,y1-(b//a)*x1,x1
```

Вычисление обратного элемента по модулю

```
def modinv(a,m):
    g,x,_=egcd(a%m,m)
    if g!=1:
        raise ValueError(f"не существует:gcd({a},{m}={g}")
    return x%m
```

Основная реализация реализующая ρ -метод.

```
def pollard_rho_dlp(p,a,b,r=None):
    if r is None:
        r=p-1

    def next_state(x,alpha,beta):
        if x%3==0:
            return (x*a)%p,(alpha+1)%r,beta
        elif x%3==1:
            return (x*b)%p,alpha,(beta+1)%r
        else:
            return (x*x)%p,(2*alpha)%r,(2*beta)%r

    x_t,alpha_t,beta_t=1,0,0
    x_h,alpha_h,beta_h=1,0,0

    for _ in range(10*int(r**0.5)+100):
        x_t,alpha_t,beta_t=next_state(x_t,alpha_t,beta_t)

        x_h,aplha_h,beta_h=next_state(x_h,alpha_h,beta_h)
        x_h,aplha_h,beta_h=next_state(x_h,alpha_h,beta_h)

        if x_t==x_h:
            A=(beta_t-beta_h)%r
            B=(alpha_h-alpha_t)%r

            if A==0:
                return None

            g=egcd(A,r)[0]
            if B%g!=0:
                return None

            A//=g
            B//=g
            r_new=r//g

            try:
                inv=modinv(A,r_new)
                x0=(B*inv)%r_new

                for k in range(g):
                    x_candidate=x0+k*r_new
                    if pow(a,x_candidate,p)==b%p:
                        return x_candidate

            except ValueError:
                return None

    return None
```

Функциональное тестирование

```
p=107
a=10
b=64
r=53
```

```
x=pollard_rho_dlp(p,a,b,r)
```

```
print(f"Решение найдено: x={x}")
```

```
Решение найдено: x=20
```

#

Выводы

1. **Теоретические знания:** Изучена задача дискретного логарифмирования и её роль в криптографии с открытым ключом (например, в протоколе Диффи–Хеллмана). Освоен принцип работы ρ-метода Полларда для решения DLP.

2. **Практические навыки:** Успешно реализован алгоритм на языке Python. Программа корректно находит дискретные логарифмы для заданных примеров и проходит верификацию.

3. **Аналитические способности:** Подтверждена эффективность ρ-метода для групп, где порядок r не слишком велик. Алгоритм демонстрирует ожидаемую сложность $O(\sqrt{r})$ и является практичным инструментом для анализа стойкости криптосистем, основанных на DLP.

Литература

1. Pollard J.M. Monte Carlo methods for index computation (mod p). Mathematics of
2. Python Software Foundation. Официальная документация Python. https://docs.python.org/3/

Приложения

Полный исходный код программ доступен в репозитории GitHub:
https://github.com/wangyao200036/cryptography-labs/tree/main/lab7