Отчет по лабораторной работе №7

Дисциплина: Математические основы защиты информации и информационной безопасности

Живцова Анна

Содержание

Список иллюстраций

Список таблиц

1 Цель работы

Изучить алгоритм Полларда для дискретного логарифмирования в конечном поле.

2 Задание

Реализовать алгоритм Полларда для дискретного логарифмирования в конечном поле.

3 Теоретическое введение

Задача дискретного логарифмирования в конечном поле – одна из первых задач, использующихся для построения криптосистем с открытым ключем. Эта задача также используется для установления сеансового ключа. Криптоскойкость данных схем основывается на вычислительной сложности решения задачи дискретного логарифмирования. Подробнее в источниках [1,2].

В данной работе будем использовать p-метод Полларда, позволяющий решить задачу дискретного логарифмирования в конечном поле порядка p, т.е. для нахождения x такого, что $a^x \equiv (mod\ p)$. Для реализации метода нужно задать сжимающую функцию на конечном множестве. Также требуется, чтобы сохранялась возможность вычислить $\log f(c)$ по известному значению $\log c$ и неизвестному $\log b$. В качестве примера такой функции в данной работе используется кусочная функция

$$f(x) = \begin{cases} ax, \text{ если } x < \frac{p}{2}, \\ bx, \text{ если } x \geq \frac{p}{2}. \end{cases}$$

4 Выполнение лабораторной работы

4.1 Программная реализация

Для реализации алгоритма дискретного логарифмирования методом Поллад-ра на языке Python была написанна следующая функция.

```
def disk_log(p, a, b, f, u, v):
    c = ((a**u)*(b**v))%p
    c_log = [u, v]
    d = c
    d_log = [u, v]
    while True:
        print(c%p, c_log, d%p, d_log)
        c, c_log_n = func(a, b, c, p)
        c_log += c_log_n
        d, d_log_n = func(a, b, d, p)
        d_log += d_log_n
        d, d_log_n = func(a, b, d, p)
        d_log += d_log_n
        if (c-d)%p == 0:
            order = find_order(a, p)
            cd_log = c_log - d_log
            r = (order - abs(cd_log[0]))
            for i in range(order):
```

```
            if (r + i*order)%cd_log[1] == 0:
                return abs((r + i*order)//cd_log[1])
```

Тут p – порядок поля, a, b – из условия задачи логарифмирования, f – сжимающая функция, u, v – начальные приближения.

Дополнительно были реализованы функции нахождения порядка элемента в поле и сжимающая функция на конечном множестве

```
def find_order(n, p):
    for i in range(1, p):
        if (n**(i))%p == 1:
            return i


def func(a, b, x, p):
    if x < p/2:
        return (a*x)%p, np.array([1, 0])
    return (b*x)%p, np.array([0, 1])
```

4.2 Проверка функциональности программы

Функциональность данной функции была протестирована в среде jupyter notebook (см. рис. 4.1). Функция действительно помогла решить задачу дискретного логарифмирования.

```
disk_log(107, 10, 64, func, 2, 2)
```

```
4 [2, 2] 4 [2, 2]
40 [3 2] 79 [4 2]
79 [4 2] 56 [5 3]
27 [4 3] 102 [6 4]
56 [5 3] 10 [7 5]
53 [5 4] 87 [8 6]
102 [6 4] 40 [9 7]
1 [6 5] 27 [10  8]
10 [7 5] 53 [11  9]
100 [8 5] 1 [12 10]
87 [8 6] 100 [14 10]

20
```

```
(10**20)%107
```

```
64
```

Рис. 4.1: Тестирование алгоритма Полладра

5 Выводы

В данной работе я изучила алгоритм Полладра для решения задачи дискретного логарифмирования в конечном поле. Также я реализовала его программно и протестировала.

Список литературы

1. Kulyabov D., Korolkova A., Gevorkyan M. Информационная безопасность компьютерных сетей: лабораторные работы. 2015.
2. Самуйлов К.Е. и др. Сети и телекоммуникации : Учебник и практикум. Издательство Юрайт, 2019. С. 1–363.