Математические основы защиты информации и информационной безопасности. Отчет по лабораторной работе №7

Шифрование гаммированием

Юдин Герман Станиславович 1132236901

Содержание

1 Цель работы **5**

2 Выполнение лабораторной работы **6**
2.1 p-метод Полларда 6

3 Выводы **9**

4 Список литературы **10**

List of Figures

2.1 main_func 7
2.2 output 8

List of Tables

1 Цель работы

Освоить на практике дискретное логарифмирование в конечном поле.

2 Выполнение лабораторной работы

Требуется реализовать:

1. Алгоритм, реализующий p-метод Полларда для задач дискретного логарифмирования

2.1 p-метод Полларда

Основные шаги:

Вход: Простое число p, числа 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 Тождественно = b (mod p), если такой показатель существует 1. Выбрать произвольные 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 тождественно = d(mod p) 3. Приравняв логарифмы для c и d, вычислить логарифм x решением сравнения по модулю r. Результат: x или “Решения нет”

Чтобы реализовать программу был написал след. код на python:

1. Функция, реализующая p-метод Полларда
2. Функция нахождения НОД
3. Расширенный алгоритм Евклида для вычисления модульного обратного элемента fig. 2.1.

```python
def pollard_p_method(p, a, b, f, r, u, v):
    # Выбор произвольных чисел u, v
    c = (a ** u * b ** v) % p
    d = c

    # Итерации метода Полларда
    while True:
        c = f(c) % p
        d = f(f(d)) % p

        # Если c = d, вычисляем логарифмы для c и d
        if c == d:
            # Вычисляем логарифм x решением сравнения по модулю r
            # Решения нет, если r и p не взаимно просты
            if gcd(r, p-1) != 1:
                return "Решения нет"

            # Вычисляем логарифм x
            x = (u - v * modinv((u - v), r) * (c - a ** u) % r) % r
            return x

# Функция вычисления наибольшего общего делителя (GCD)
def gcd(a, b):
    while b:
        a, b = b, a % b
    return a

# Расширенный алгоритм Евклида для вычисления модульного обратного элемента
def modinv(a, m):
    m0, x0, x1 = m, 0, 1
    while a > 1:
        q = a // m
        m, a = a % m, m
        x0, x1 = x1 - q * x0, x0
    return x1 + m0 if x1 < 0 else x1
```

Figure 2.1: main_func

Выходные значения программы fig. 2.2.

```python
# Пример использования
p = 107
a = 10
b = 64
r = 53
u = 2
v = 2

# Определение функции f
def f(c):
    if c < r:
        return (10 * c) % p
    else:
        return (64 * c) % p

result = pollard_p_method(p, a, b, f, r, u, v)
print("Решение:", result)

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS F:\учеба 5 курс\информационная безопасность> python .\Lab7_log\lab7_logs.py
Решение: Решения нет
PS F:\учеба 5 курс\информационная безопасность>
```

Figure 2.2: output

3 Выводы

В результате выполнения работы я освоил на практике дискретное логарифмирование в конечном поле.

4 Список литературы

1. Методические материалы курса