Лабораторная Работа No 7

Математические Основы Защиты Информации и Информационной Безопасности

Хосе Фернандо Леон Атупанья | НФИмд-01-24

Содержание

1. Цель работы

Ознакомиться с темой дискретного логарифмирования в конечном поле, используя материал, представленный в лабораторной работе № 7, и используя концепции, представленные в предыдущих работах, такие как модули, максимальный общий делитель и шифрование.

2. Выполнение лабораторной работы - Дискретное логарифмирования в конечном поле

Алгоритм, реализующий р-метод Полладла для задач дискретного логарифмирования

В этой отчете следующий код реализует р-метод Полларда для задач дискретного логарифмирования. Р-метод (rho-метод Полларда) для решения задачи дискретного логарифмирования является вероятностным алгоритмом. Он эффективен для нахождения показателя степени x в $a^x \in b (\mathrm{mod}\ p)$, где p - простое число, a - генератор циклической группы, а b - целевое значение. Ниже приводится объяснение алгоритма.

Для начала мы вычисляем экспоненциальную модульную следующим образом: сначала мы инициализируем нашу переменную "результат" значением 1 и с помощью функции while вычисляем и проверяем, что наше входное число четное или нечетное. В зависимости от этого значение нашей переменной "результат" будет меняться

```
# Function to compute modular exponentiation
function mod_exp(base, exp, mod)
    result = 1
    while exp > 0
        if exp % 2 == 1
            result = (result * base) % mod
        end
        base = (base * base) % mod
        exp ÷= 2
    end
    return result
end
```

Функция pollard_rho_dlog:

Инициализирует c и d случайным значением u, v. Обновляет c и d, используя предоставленную функцию отображения f, отслеживая изменения в u, v. Обнаруживает коллизию (c==d). Решает для x путем приравнивания логарифмов и использования модульной арифметики.

```julia
# Function implementing Pollard's rho method
function pollard_rho_dlog(p, a, b, r, f)
    # Step 1: Choose random u, v and initialize c and d
    u, v = 2, 2  # Example initialization, these can be random
    c = (mod_exp(a, u, p) * mod_exp(b, v, p)) % p
    d = c
    u_c, v_c = u, v
    u_d, v_d = u, v
```

Функция р-метода Полларда, эта функция реализует основной алгоритм нахождения x таким образом, что:

Основная функция pollards_p_method(p, a, b, r, f) принимает в качестве входных данных: p: число для разложения на множители. a: Начальное значение для алгоритма b: Начальное значение для алгоритма f: Псевдослучайная функция сжатия.

u и v - это показатели для a и b соответственно, изначально установленные равными 2.

d также инициализируется как c.

```julia
    # Step 2: Update c and d using the function f and track log
    while true
        # Update c
        if c < r
            u_c = (u_c + 1) % r
        else
            v_c = (v_c + 1) % r
        end
        c = f(c, p, r)

        # Update d twice
        for _ in 1:2
            if d < r
                u_d = (u_d + 1) % r
            else
                v_d = (v_d + 1) % r
            end
            d = f(d, p, r)
        end
```

Алгоритм использует метод "черепахи и зайца" для поиска коллизии. Обнаружение коллизий: Когда c = d, обнаруживается петля, сигнализирующая о возможном решении.

```
        # Check for collision
        if c == d
            break
        end
    end

    # Step 3: Solve for x
    numerator = (u_c - u_d) % r
    denominator = (v_d - v_c) % r

    # Solve numerator / denominator mod r using modular inverse
    inv_denominator = mod_exp(denominator, r - 2, r)  # Fermat'
    x = (numerator * inv_denominator) % r

    # Verify result
    if mod_exp(a, x, p) == b
        return x
    else
        return "No solution"
```

Остаток от деления по модулю вычисляется с использованием модулярной функции, обратной с (согласно Малой теореме Ферма).Решение x проверяется путем проверки того, что a^x = b(modp). Вычисляет дискретный логарифм x, который выводится в качестве результата.

OUTPUT:

```
# Example input and function
p = 107
a = 10
b = 64
r = 53

# Define the function f
function f(c, p, r)
    return c < r ? (10 * c) % p : (64 * c) % p
end

# Solve using Pollard's rho
x = pollard_rho_dlog(p, a, b, r, f)
println("Discrete logarithm x = $x")
```

3. Выводы

В этом упражнении p-метод Полларда был реализован в Julia для разложения целых чисел на множители. Алгоритм успешно продемонстрировал свою способность находить нетривиальные делители составных чисел, используя псевдослучайные итеративные обновления и свойства наибольшего общего делителя. Используя пример с p = 107, a = 10, b = 64, r = 53, алгоритм определил как нетривиальный фактор, подтверждающий его эффективность.

Это упражнение подчеркивает практическую полезность алгоритмов теории чисел в вычислительной математике, криптографии и решении задач. Кроме того, оно демонстрирует простоту реализации передовых математических методов в Julia, подчеркивая пригодность языка для решения математических и алгоритмических задач.