marp: true css: custom-theme.css title: "Лабораторная работа №6"на простоту" subtitle: "Дисциплина: Математические основы защиты информации и информационной безопасности" author: Хосе Фернандо Леон Атупанья, НФИмд-01-24, 1032249918 institute: Российский Университет Дружбы Народов, Москва, Россия date: 26 октября 2024

Лабораторная работа № 7

Дисциплина: Математические основы защиты информации и информационной безопасности

Тема: Дискретное логарифмирование в конечном поле

Студент: Леон Фернандо Хосе Фернандо

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.

Задание

1. Реализовать алгоритм программно
2. Получить у преподователя задание, содержащее числа p, a, b и вычислить логарифм.

2. Выполнение лабораторной работы

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

P-метод (rho-метод Полларда) для решения задачи дискретного логарифмирования является вероятностным алгоритмом. Он эффективен для нахождения показателя степени x в a ^x ∈b(mod p), где p - простое число, a - генератор циклической группы, a b - целевое значение. Ниже приводится объяснение алгоритма.

1. Вычислить модульное возведение в степень

```
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
```

---

1. Функция, реализующая метод р Полларда (1/4)

```
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

    # Step 2: Update c and d using the function f and track logs
    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)
```

---

1. Функция, реализующая метод р Полларда (2/4)

```
       # 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

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

1. Функция, реализующая метод p Полларда (3/4)

```
    # 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's little theorem
    x = (numerator * inv_denominator) % r

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

1. Функция, реализующая метод p Полларда (4/4)

```
# 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")
```

Вывод

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.