Лабораторная работа №7

Математические основы защиты информации и информационной безопасности

Махорин И. С.

2025

Российский университет дружбы народов имени Патриса Лумумбы, Москва, Россия

Докладчик

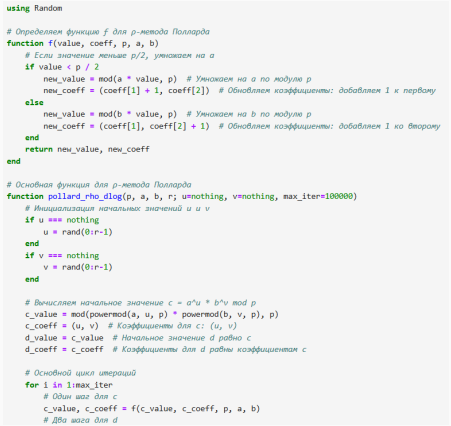
- Махорин Иван Сергеевич
- Студент группы НФИмд-02-25
- Студ. билет 1032259380
- Российский университет дружбы народов имени Патриса Лумумбы

Цель лабораторной работы

- Изучить алгоритм дискретного логарифмирования в конечном поле и научиться его реализовывать.

Выполнение лабораторной работы

Реализация алгоритма, реализующего P-Метод Полларда для задач дискретного логарифмирования



```julia
using Random

# Определяем функцию f для ρ-метода Полларда
function f(value, coeff, p, a, b)
    # Если значение меньше p/2, умножаем на a
    if value < p / 2
        new_value = mod(a * value, p)  # Умножаем на a по модулю p
        new_coeff = (coeff[1] + 1, coeff[2])  # Обновляем коэффициенты: добавляем 1 к первому
    else
        new_value = mod(b * value, p)  # Умножаем на b по модулю p
        new_coeff = (coeff[1], coeff[2] + 1)  # Обновляем коэффициенты: добавляем 1 ко второму
    end
    return new_value, new_coeff
end

# Основная функция для ρ-метода Полларда
function pollard_rho_dlog(p, a, b, r; u=nothing, v=nothing, max_iter=100000)
    # Инициализация начальных значений u и v
    if u === nothing
        u = rand(0:r-1)
    end
    if v === nothing
        v = rand(0:r-1)
    end

    # Вычисляем начальное значение c = a^u * b^v mod p
    c_value = mod(powermod(a, u, p) * powermod(b, v, p), p)
    c_coeff = (u, v)  # Коэффициенты для c: (u, v)
    d_value = c_value  # Начальное значение d равно c
    d_coeff = c_coeff  # Коэффициенты для d равны коэффициентам c

    # Основной цикл итераций
    for i in 1:max_iter
        # Один шаг для c
        c_value, c_coeff = f(c_value, c_coeff, p, a, b)
        # Два шага для d
```

Рис. 1: Реализация алгоритма, реализующего P-Метод Полларда для задач дискретного логарифмирования

Реализация алгоритма, реализующего P-Метод Полларда для задач дискретного логарифмирования

```
        d_value, d_coeff = f(d_value, d_coeff, p, a, b)
        d_value, d_coeff = f(d_value, d_coeff, p, a, b)

        # Проверяем коллизию
        if c_value == d_value
            A1, B1 = c_coeff
            A2, B2 = d_coeff
            # Формируем уравнение: (B1 - B2)*x ≡ (A2 - A1) mod r
            left = mod(B1 - B2, r)
            right = mod(A2 - A1, r)

            # Если left = 0, уравнение вырождено
            if left == 0
                if right == 0
                    error("Уравнение имеет бесконечно много решений")
                else
                    error("Уравнение не имеет решений")
                end
            end

            # Решаем линейное уравнение
            g, inv_left, _ = gcdx(left, r)
            if right % g != 0
                error("Уравнение не имеет решений")
            else
                r_prime = div(r, g)
                x0 = mod(inv_left * div(right, g), r_prime)
                # Проверяем все возможные решения
                for t in 0:g-1
                    x_candidate = mod(x0 + t * r_prime, r)
                    if powermod(a, x_candidate, p) == b
                        return x_candidate
                    end
                end
                error("Найдены решения уравнения, но ни одно не является логарифмом")
            end
        end
```

Рис. 2: Реализация алгоритма, реализующего P-Метод Полларда для задач дискретного логарифмирования

```
        end
    end
    error("Коллизия не найдена за $max_iter итераций")
end

# Функция для проверки корректности вычисленного логарифма
function verify_dlog(p, a, b, x)
    return powermod(a, x, p) == b
end
```

Рис. 3: Реализация алгоритма, реализующего P-Метод Полларда для задач дискретного логарифмирования

Проверка

```
# Пример использования
p = 107
a = 10
b = 64
r = 53

# Вычисляем логарифм
x = pollard_rho_dlog(p, a, b, r; u=2, v=2)
println("Найденный x: $x")

# Проверяем корректность
if verify_dlog(p, a, b, x)
    println("Проверка пройдена: $a^$x ≡ $b (mod $p)")
else
    println("Ошибка: $a^$x ≠ $b (mod $p)")
end
```

```
Найденный x: 20
Проверка пройдена: 10^20 ≡ 64 (mod 107)
```

Рис. 4: Проверка

Вывод

Вывод

- В ходе выполнения лабораторной работы был изучен алгоритм дискретного логарифмирования в конечном поле, а также написан его алгоритм на языке Julia.

Список литературы. Библиография

Список литературы. Библиография

[1] Julia: https://docs.julialang.org/en/v1/