Отчет по лабораторной работе №7

дисциплина: Математические основы защиты информации и информационной безопасности

Морозова Ульяна

Содержание

1 Цель работы

Целью работы является изучение алгоритм разложение чисел на множителей и реализация его на языке Julia.

2 Выполнение лабораторной работы

2.1 Алгоритм p-метода Полларда для задачи дискретного логарифмирования

Алгоритм ρ-Полларда (Pollard's rho) — это вероятностный алгоритм для вычисления дискретного логарифма. Он особенно эффективен, когда порядок циклической группы является составным числом с небольшими простыми делителями, но работает и в общем случае быстрее, чем алгоритм перебора (Baby-step Giant-step), потребляя при этом значительно меньше памяти.

Ниже представлен код на языке Julia, реализующий этот алгоритм для мультипликативной группы кольца вычетов по модулю простого числа p.

```
function extended_gcd(a::BigInt, b::BigInt)
    if a == 0
        return b, 0, 1
    else
        g, y, x = extended_gcd(b % a, a)
        return g, x - (b ÷ a) * y, y
    end
end

function pollard_rho_dlp(alpha::Union{Int, BigInt}, beta::Union{Int, BigInt}, p::Union
    alpha, beta, p = BigInt(alpha), BigInt(beta), BigInt(p)
```

```
n = p - 1
function step(x, a, b)
    mode = x % 3
    if mode == 0
        # S0: x -> x^2
        x_new = powermod(x, 2, p)
        a_new = (a * 2) % n
        b_new = (b * 2) % n
    elseif mode == 1
        # S1: x -> x * alpha
        x_new = (x * alpha) % p
        a_new = (a + 1) % n
        b_new = b
    else
        # S2: x -> x * beta
        x_new = (x * beta) % p
        a_new = a
        b_new = (b + 1) % n
    end
    return x_new, a_new, b_new
end

x, a, b = BigInt(1), BigInt(0), BigInt(0)
X, A, B = x, a, b


for i in 1:p
    x, a, b = step(x, a, b)
```

```
X, A, B = step(X, A, B)
X, A, B = step(X, A, B)

if x == X

    r = (b - B) % n
    m = (A - a) % n



    if r < 0 r += n end
    if m < 0 m += n end



    d, u, v = extended_gcd(r, n)

    if m % d != 0
        return nothing
    end

    # Частное решение
    w = (u * (m ÷ d)) % (n ÷ d)
    if w < 0 w += (n ÷ d) end



    for k in 0:(d-1)
        candidate_x = w + k * (n ÷ d)
        if powermod(alpha, candidate_x, p) == beta
            return candidate_x
```

```
                end
            end

            return nothing
        end
    end
    return nothing
end

# --- Пример использования ---

# 2^x = 22 (mod 29) -> Ответ должен быть 13, так как 2^13 = 8192, 8192 % 29 = 22
alpha = 2
beta = 22
prime = 29

println("Задача: $alpha^x = $beta (mod $prime)")
result = pollard_rho_dlp(alpha, beta, prime)

if result !== nothing
    println("Дискретный логарифм x: $result")
    println("Проверка: $(powermod(alpha, result, prime)) == $beta")
else
    println("Не удалось найти решение.")
end
```

И результат его работы

```
Задача: 2^x = 22 (mod 29)
Дискретный логарифм x: 26
Проверка: 22 == 22
```

3 Выводы

Мы изучили работу алгоритма, а также реализовали его на языке Julia.