Politechnika Wrocławska

Prowadzący : Prof. Janusz Biernat

Wykonawcy: Jarosław Sularz 218316

Kacper Szymula 226215

Termin: Czwartek 16.30

Organizacja i Architektura Komputerów

Projekt

Temat: Obliczanie logarytmu dyskretnego

1. Wstęp

Problem logarytmu dyskretnego został wykorzystany przez Diffiego i Hellmana do stworzenia protokołu bezpiecznej wymiany klucza w komunikacji sieciowej. Dzięki jego zastosowaniu możliwe jest ustalenie klucza szyfrującego w sposób, który uniemożliwia jego przechwycenie lub podsłuchanie. Może jednak zacznę od podania odpowiedniej definicji. „Niech $Fp * = (Z/pZ) * = \{1, 2, \ldots, p-1\}$ będzie grupą multiplikatywną liczb całkowitych modulo liczba pierwsza p (…). Niech $g \in Fp *$ będzie ustalonym elementem (naszą „podstawą"). Problemem logarytmu dyskretnego w $Fp *$ przy podstawie g nazywamy zadanie wyznaczenia dla danego $y \in Fp *$ takiej liczby naturalnej x, że $y = gx \, mod \, p$ (o ile takie x istnieje — w przeciwnym wypadku powinniśmy otrzymać jako wynik stwierdzenie, że y nie należy do podgrupy generowanej przez g)" .

Do obliczenia algorytmu dyskretnego wykorzystałem algorytm Brute Force, który polega na sekwencyjnym generowaniu wartości g^x, która jest sekwencyjnie sprawdzana z bazową wartością y.

2. Kod programu

Kod assemblerowy obliczający logarytm dyskretny oraz sprawdzający czy liczby a i m są pierwsze.

```
.bss

.data
a: .int 0, 0
b: .int 0, 0
m: .int 0, 0
bmodm: .int 0, 0
ak: .int 0, 0

.text



#funkcja sprawdza pierwszosc liczb
#w rej.
#rax = x
#funkcja zwraca wynik w rax, 0 - nie pierwsza
prime:
            #licznik c w rsi
            mov $2, %rsi
            mov %rax,%rbx          #kopia
prime0:
            cmp %rbx,%rsi
            jge prime1
            xor %rdx,%rdx
            div %rsi
            cmp $0, %rdx     #porownanie reszyt z dzislenia
            je prime2
            inc %rsi
            mov %rbx,%rax
            jmp prime0
prime2:
            mov $0, %rax
            jmp prime_ret
prime1:
            mov $1, %rax
prime_ret:
            ret





.globl discreteLogarithm
.type discreteLogarithm, @function

discreteLogarithm:
        pushq %rbp
        movq %rsp, %rbp

            #rdi = a
```

```
            #rsi = b
            #rdx = m
            mov %rdi, a
            mov %rsi, b
            mov %rdx, m

            #najpierw sprwdzamy pierwszoc a i m
            mov a, %rax
            call prime
            cmp $1, %rax
            jne dl_end0
            mov m, %rax
            call prime
            cmp $1, %rax
            jne dl_end0

            #jezlei a i m sa pierwsze
            xor %rdx,%rdx
            mov b, %rax
            divq m
            #reszta w rdx
            mov %rdx, bmodm

            #rsi to licznik k
            xor %rsi, %rsi
            movq $1, ak
dl0:
            xor %rdx,%rdx
            mov ak, %rax
            divq m
            #reszta w rdx = akmodm
            cmp %rdx, bmodm
            je dl_end1

            #jak nie rowne to szukamy dalej
            mov ak,%rax
            mulq a                #a^k
            mov %rax, ak
            inc %rsi
            jmp dl0

dl_end1:
            mov %rsi, %rax
            jmp dl_ret
dl_end0:
            mov $-1, %rax
dl_ret:
            #return ans, wynik w rax

        movq %rbp, %rsp
        popq %rbp
        ret
```

Funkcja main() napisana w języku C.

```
      // C program to calculate discrete logarithm
#include <stdio.h>


// Function to calculate k for given a, b, m
extern long long int discreteLogarithm(int a, int b, int m);

int main()
{
    //     int a = 59, b = 3, m = 1307;
      printf("a = ");
      scanf("%i", &a);
      printf("b = ");
      scanf("%i", &b);
      printf("m = ");
      scanf("%i", &m);
      printf("Wynik logarytmu dyskretnego = %lli\n",
      discreteLogarithm(a, b, m));
      return 0;
}
```

Wynikiem działania programu jest znaleziony logarytm lub -1.

Wartości -1 dostajemy jeżeli a oraz m nie są pierwsze lub nie znaleziono rozwiązania dla podanych danych.

3. Wnioski

Działanie programu przetestowałem dla 20 małych wartości a oraz m. Wszystkie wyniki były poprawne. Złożoność obliczeniowa zastosowanego algorytmu jest O(n).

Istnieje możliwość poprawienia złożoności algorytmu do O(sqrt(n)*log(b)) stosując algorytm baby-step giant-step. Aby wykorzystać ten algorytm potrzebny jest kontener na dane który pozwala wyszukać wartość dla podanego klucza. W językach niskiego poziomu jest to skomplikowane działanie dlatego zdecydowałem się na użycie metody siłowej.

4. Bibliografia

- „Podstawy Kryptografii”, Marcin Karbowski, s 70-72
- https://www.khanacademy.org/computing/computer-science/cryptography/modern-crypt/v/discrete-logarithm-problem
- https://www.geeksforgeeks.org/discrete-logarithm-find-integer-k-ak-congruent-modulo-b