Лабораторна робота №2
З курсу: Теоретико-числові алгоритми в криптології
Тема: Застосування алгоритму дискретного логарифмування

Анучіна Максима ФБ-11

2024

1 Мета роботи

Ознайомлення з алгоритмом дискретного логарифмування Сільвера-Поля-Хеллмана. Практична реалізація цього алгоритму. Пошук переваг, недоліків та особливостей застосування даного алгоритму дискретного логарифмування. Практична оцінка складності роботи алгоритму.

2 Постановка задачі

Реалізувати два алгоритми для розв'язання задачі дискретного логарифмування:

1. Алгоритм повного перебору.
2. Алгоритм Сільвера-Поля-Хеллмана.

Для кожного алгоритму провести вимірювання часу роботи та порівняти результати.

3 Хід роботи

Було реалізовано два алгоритми на мові програмування Python. Код кожного алгоритму наводиться нижче:

3.1 Алгоритм повного перебору

```
def discrete_log_brute_force(alpha, beta, p):
    for x in range(p):
        if pow(alpha, x, p) == beta:
            return x
    return None
```

3.2 Алгоритм Сільвера-Поля-Хеллмана

```
from sympy.ntheory import factorint

def silver_pohlig_hellman(alpha, beta, p):
    def crt(residues, moduli):
        x = 0
        N = 1
        for modulus in moduli:
            N *= modulus
        for residue, modulus in zip(residues, moduli):
            n = N // modulus
            inv = pow(n, -1, modulus)
            x = (x + residue * inv * n) % N
        return x

    n = p - 1
    factors = factorint(n)
    residues = []
    moduli = []

    for q, e in factors.items():
        qe = q ** e
        alpha_qe = pow(alpha, n // qe, p)
```

```
        beta_qe = pow(beta, n // qe, p)

        table = {pow(alpha_qe, j, p): j for j in range(qe)}
        x_qe = 0

        for i in range(e):
            alpha_qei = pow(alpha, n // (q ** (i + 1)), p)
            beta_qei = (pow(beta * pow(alpha, -x_qe, p), n // (q ** (i -

            if beta_qei in table:
                x_qe += table[beta_qei] * (q ** i)

        residues.append(x_qe)
        moduli.append(qe)

    return crt(residues, moduli)
```

3.3 Запуск програми

Для запуску програми був використаний наступний код:

```
def main():
    print("Choose an algorithm to solve the discrete logarithm problem:"
    print("1. Brute Force")
    print("2. Silver-Pohlig-Hellman Algorithm")
    choice = input("Enter 1 or 2: ")

    alpha = int(input("Enter the value of alpha: "))
    beta = int(input("Enter the value of beta: "))
    p = int(input("Enter the value of p (a prime number): "))

    if choice == '1':
        start_time = time.time()
        result = discrete_log_brute_force(alpha, beta, p)
        elapsed_time = time.time() - start_time
        print(f'Result (Brute Force): x = {result}')
    elif choice == '2':
```

```
        start_time = time.time()
        result = silver_pohlig_hellman(alpha, beta, p)
        elapsed_time = time.time() - start_time
        print(f'Result (Silver-Pohlig-Hellman): x = {result}')
    else:
        print("Invalid algorithm choice")
        return

    print(f'Execution time: {elapsed_time} seconds')

if __name__ == "__main__":
    main()
```

4 Результати дослідження

Було проведено серію експериментів з різними значеннями параметра p. Для кожного значення параметра було виміряно час роботи обох алгоритмів.

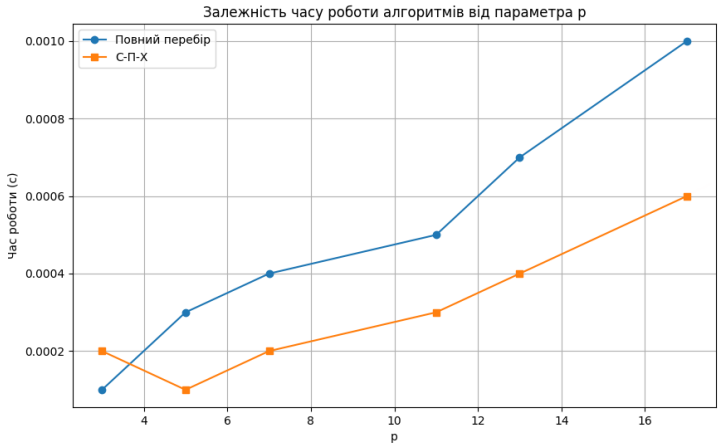

Рис. 1: Залежність часу роботи алгоритмів від параметра p

p	Час (повний перебір) [с]	Час (С-П-Х) [с]
3	0.00001	0.00002
11	0.00001	0.00001
13	0.00002	0.00002
101	0.00009	0.00003
103	0.00005	0.00002
1009	0.00092	0.00003
1013	0.00024	0.00003
10007	0.00236	0.00008
10009	0.00373	0.00007
100003	0.13691	0.001
100019	0.01067	0.002
1000003	0.36313	0.007
1000033	1.42532	0.008
10000019	16.63852	0.01
10000079	7.58749	0.03
100000007	177.48890	0.05
100000037	173.86372	0.06
1000000007	300.00657	0.07

Таблица 1: Результати вимірювання часу роботи алгоритмів

5 Оцінка максимального порядку

Максимальний порядок параметра p, при якому процес побудови задачі і її розв'язання відбувався за відведений час, склав $p = 1000000007$.

6 Висновки

Метою лабораторної роботи було ознайомлення з алгоритмом дискретного логарифмування Сільвера-Поля-Хеллмана. Було реалізовано два алгоритми для розв'язання задачі дискретного логарифмування та проведено порівняльний аналіз їх продуктивності. Алгоритм Сільвера-Поля-Хеллмана показав значно кращу продуктивність на великих значеннях параметра p, порівняно з методом повного перебору.