Министерство науки и высшего образования Российской Федерации
Санкт-Петербургский Политехнический Университет Петра Великого
—
Институт компьютерных наук и кибербезопасности
Высшая школа кибербезопасности

ЛАБОРАТОРНАЯ РАБОТА № 5

«Разложение числа на множители на эллиптической кривой»

по дисциплине «Криптографические методы защиты информации»

Выполнил
студент гр. 5151004/90101 Кондачков Е.Д.

Преподаватель
Ассистент Ярмак А.В.

Санкт-Петербург
2023

Содержание

1. Цель работы

Реализация метода разложения числа на множители с использованием эллиптических кривых.

2. Задача работы

Получить у преподавателя вариант задания и разработать программу П-1, которая находит разложение составного числа n на эллиптической кривой. В качестве входных данных программа П-1 должна принимать число n, размер m базы D; на выходе – возвращать нетривиальный делитель d числа n.

3. Ход работы

В ходе данной лабораторной работы необходимо было реализовать программу П-1, которая находит разложение составного числа n на эллиптической кривой.

Сначала программа строит базу D размера m, которая состоит из простых чисел, начиная с 2 и пока количество не достигнет m. Далее выбирается точка Q, после чего строится случайная эллиптическая кривая E(Z/nZ). Параметры эллиптической кривой и координаты точки при n=661643 представлены ниже:

Q_x = 46312

Q_y = 501345

A = 2

B = 598638

Время варьируется в зависимости от того, какие параметры генерируются для ЭК и какие координаты для точки. Ниже будет приведена таблица с примерным временем и итерациями:

Таблица 1 – Результаты проведенных тестов программы.

n	Размер базы	Время, с	Итерации (сложение точек)	Простые делители
661643	5	1.37	114	541, 1223
661643	10	0.1	89	
661643	20	0.6	156	
1315237	5	0.4	70	31, 29, 7, 11, 19
1315237	10	0.42	134	
1315237	20	0.62	89	
6519439	20	6.42	110	199, 181, 181
7825781	20	7.19	99	3449, 2269

Что удалось выяснить точно, так это то, что факторизация на прямую зависит от того, как быстро будет построена эллиптическая кривая и на сколько удачно.

4. Ответы на контрольные вопросы

1. Как зависит сложность разложения составного числа заданной длины методом эллиптических кривых от числа различных простых делителей числа n?

Пусть составное число представимо в виде произведения k простых чисел $n = q_1 \dots q_k$. Тогда при увеличении числа простых делителей их длина будет расти и станет примерно одинаковой. Алгоритм Ленстры зависит от длины минимального простого делителя, поэтому при увеличении числа простых делителей заданного числа будет расти и сложность разложения.

2. Сравните сложность разложения на эллиптической кривой составного числа вида $n = p^2 q$, где p, q – различные простые числа, и составного числа такой же длины, состоящего из двух различных простых делителей.

Пусть есть два числа следующего вида:

$$n_1 = p_1^2 q_1,$$
$$n_2 = p_2 q_2,$$

Причем эти числа обладают одинаковой длиной. Пусть длины множителей q_1 и q_2 − равны, тогда длины множителей p_1^2 и p_2 будут также равными. Следовательно, будет справедливо следующее неравенство относительно длин множителей: $|p_1| < |p_2|$. Так как при использовании алгоритма Ленстры быстрее раскладываются те числа, которые имеют простой делитель меньшего размера, то в данном случае сложность разложения n_1 будет меньше сложности разложения n_2.

5. Вывод

В ходе лабораторной работы был изучен алгоритм факторизации числа на ЭК. Он показал себя довольно хорошо, но вот прежде, чем начать факторизовать число, необходимо построить саму ЭК. Что занимает достаточно много времени. Поэтому не удалось оценить работу алгоритма, но я уверен, что он хороший.

Приложение А

Листинг программы

```
from Crypto.Random import get_random_bytes
from Crypto.Util.number import getPrime, inverse, GCD
import random
from random import randint
import time
from math import log
from sympy import isprime
import numpy as np

iterations = 0

# Сложение точек на ЭК
def get_points_sum(x1, y1, x2, y2, A, p, n):

    L = 0

    # Если складываем с бесконечно удаленной точкой
    if x1 == 0 and y1 == 0:
        return x2, y2, 0
    elif x2 == 0 and y2 == 0:
        return x1, y1, 0
    # Если складываем P = (a , b) и P = (a, -b)
    elif x1 == x2 and y1 == -y2:
        return 0, 0, 0 # бесконечно удаленная точка
    # Если P = Q
    elif x1 == x2 and y1 == y2:
        d = GCD(p, 2 * y1)
        if(d > 1 and d < n):
            return x2, y2, d
        L = ((3 * (x1**2) + A) * inverse(2 * y1, p)) % p
    # Если P != Q
    else:
        d = GCD(p, x2 - x1)
        if(d > 1 and d < n):
            return x2, y2, d
        L = ((y2 - y1) * inverse((x2 - x1), p)) % p
```

```
    x3 = (L**2 - x1 - x2) % p
    y3 = (L * (x1 - x3) - y1) % p

    return x3, y3, d

# Создание двоичного вектора
def get_binary_vector(x):

    lst = []

    while(x != 0):
        lst.append(x % 2)
        x //= 2

    vctr = np.array(lst[::-1])

    return vctr

# Бинарное умножение точки на число
def multiply_binary(P_x, P_y, k, A, n):

    if(k == 0):
        return 0, 0 # бесконечно удаленная точка
    elif(k == 1):
        return P_x, P_y

    a = get_binary_vector(k)

    Q_x, Q_y = 0, 0
    global iterations

    for i in a:
        Q_x, Q_y, d = get_points_sum(Q_x, Q_y, Q_x, Q_y, A, n)
        iterations += 1

        if d > 1 and d < n:
            return Q_x, Q_y, d
```

```
        if(i == 1):
            Q_x, Q_y, d = get_points_sum(Q_x, Q_y, P_x, P_y, A, n)
            iterations += 1

        if d > 1 and d < n:
            return Q_x, Q_y, d

    return Q_x, Q_y, d

# Создание базы разложения
def get_base(m):

    a = []
    a.append(2)
    count = 1
    num = 3

    while True:
        if(isprime(num)):
            a.append(num)
            count += 1
        num += 2

        if(count == m):
            break

    return a

# Факторизация
def get_prime_divisors(n, m):

    D = get_base(m)

    while True:

        # Генерируем координаты точки
        Q_x = randint(1, n)
        Q_y = randint(1, n)
```

```
            # Генерируем коэффициенты
            while True:
                A = randint(-2, 3)
                B = randint(1, n)

                k = (4 * pow(A, 3, n) + 27 * pow(B, 2, n)) % n
                if(k != 0 and pow(Q_y, 2, n) == (pow(Q_x, 3, n) + A * Q_x + B)
% n):
                    break

            i = 0
            Q_xi, Q_yi = Q_x, Q_y

            while i < m:
                ai = int(0.5 * (log(n)/log(D[i]))) # натуральный логарифм
                j = 0

                while j <= ai:
                    Q_xi, Q_yi, d = multiply_binary(Q_xi, Q_yi, D[i], A, n)
                    j += 1

                    if d > 1 and d < n:
                        if(isprime(d) != True):
                            d = get_prime_divisors(d, m)
                        return d
                i += 1

    def get_multipliers(number, base):
        divisors = []
        while(isprime(number) == False):
           d = get_prime_divisors(number, base)
           number //= d
           divisors.append(d)
        divisors.append(number)

    def main():
        while(True):
            n = int(input('Enter your fighter\'s number (number):'))
            m = int(input('Enter the arena number (base):'))
```

```
        start = time.perf_counter()
        divisors, iterations = get_multipliers(n, m)
        stop = time.perf_counter()

        print(f'Achievements received (prime divisors): {divisors}\n\
Fight time: {stop - start} second\n\
Number of rounds: {iterations}')

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