МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Московский Авиационный Институт»

(Национальный Исследовательский Университет)

Институт: №8 «Информационные технологии и прикладная математика» Кафедра: 806 «Вычислительная математика и программирование»

Лабораторная работа № 6 по курсу

«Криптография»

Группа: М8О-306Б-20

Студент: И. П. Попов

Преподаватель: А. В. Борисов

Оценка:

Дата:

Москва, 2022

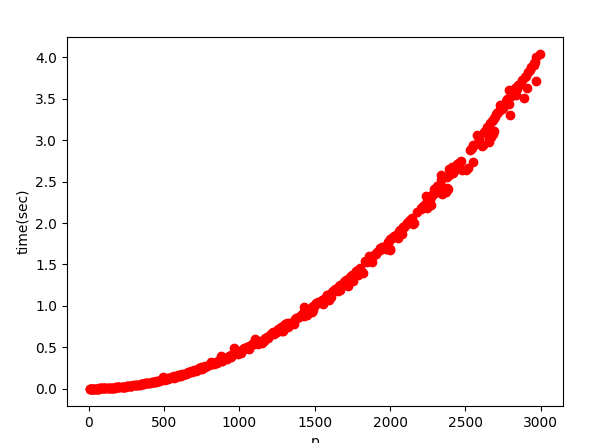
# **Задача**

Подобрать такую эллиптическую кривую, порядок точки которой полным перебором находится за 10 минут на ПК. Упомянуть в отчёте результаты замеров работы программы, характеристики вычислителя. Также указать какие алгоритмы и/или теоремы существуют для облегчения и ускорения решения задачи полного перебора. Рассмотреть для случая конечного простого поля Z\_p.

# 

# **Метод решения**

Я взял эллиптическую кривую *y*2 = *x*3 + *ax* + *b*, выбрал случайным образом коэффициенты *a* и *b*. С помощью решета Эратосфена сформировал массив простых чисел до 3000 и посмотрел, сколько времени для разных *p* занимает поиск всех точек кривой и поиск порядка случайно выбранной точки. Получил такой результат:



Из данного соотношения предположил, что *p* должно быть около 37000.

Поиск всех точек занимает много времени, это происходит из-за полного перебора (Θ(*p*2)). Далее ищем порядок случайно выбранной из найденных точки. Складываем ее с самой собой до тех пор, пока не получим нулевую точку. Количество операций сложения и будет являться искомым порядком.

# **Исходный код**

import random

import time

import matplotlib.pyplot as plt

import numpy as np

A = random.randint(1000000000, 10000000000)

B = random.randint(1000000000, 10000000000)

def print\_curve():

print("y^2 = x^3 + {0} \* x + {1} (mod {2})".format(A % p, B % p, p))

def elliptic\_curve(x, y, p):

return (y \*\* 2) % p == (x \*\* 3 + (A % p) \* x + (B % p)) % p

def find\_points():

points = []

for x in range(p):

for y in range(p):

if elliptic\_curve(x, y, p):

points.append((x, y))

return points

def extended\_euclidean\_algorithm(a, b):

s, old\_s = 0, 1

t, old\_t = 1, 0

r, old\_r = b, a

while r != 0:

quotient = old\_r // r

old\_r, r = r, old\_r - quotient \* r

old\_s, s = s, old\_s - quotient \* s

old\_t, t = t, old\_t - quotient \* t

return old\_r, old\_s, old\_t

def inverse\_of(n, p):

gcd, x, y = extended\_euclidean\_algorithm(n, p)

assert (n \* x + p \* y) % p == gcd

if gcd != 1:

raise ValueError(

'{} has no multiplicative inverse '

'modulo {}'.format(n, p))

else:

return x % p

def add\_points(p1, p2, p):

x1, y1 = p1[0], p1[1]

x2, y2 = p2[0], p2[1]

if p1 == (0, 0):

return p2

elif p2 == (0, 0):

return p1

elif x1 == x2 and y1 != y2:

return (0, 0)

if p1 == p2:

m = ((3 \* x1 \*\* 2 + (A % p)) \* inverse\_of(2 \* y1, p)) % p

else:

m = ((y1 - y2) \* inverse\_of(x1 - x2, p)) % p

x3 = (m \*\* 2 - x1 - x2) % p

y3 = (y1 + m \* (x3 - x1)) % p

return [x3, -y3 % p]

def point\_order(point, p):

i = 1

new\_point = add\_points(point, point, p)

while new\_point != (0, 0):

new\_point = add\_points(new\_point, point, p)

i += 1

return i

def sieve(n):

primes = 2 \* [False] + (n - 1) \* [True]

for i in range(2, int(n \*\* 0.5 + 1.5)):

for j in range(i \* i, n + 1, i):

primes[j] = False

return [prime for prime, checked in enumerate(primes) if checked]

if \_\_name\_\_ == '\_\_main\_\_':

primes = sieve(37000)

p = primes[-1]

start = time.time()

points = find\_points()

points\_num = len(points)

print\_curve()

print("Elliptic curve group order = {0}".format(points\_num))

point = random.choice(points)

print("Order of point {0}: {1}".format(point, point\_order(point, p)))

print("Time: {0}".format(time.time() - start))

**Консоль**

y^2 = x^3 + 10782 \* x + 8823 (mod 36997)

Elliptic curve group order = 36845

Order of point (20317, 33466): 6140

Time: 966.7980222702026

# 

# Код для построения графика выше

import matplotlib.pyplot as plt

import numpy as np

from math import exp

def F(x):

return exp(-x) / x

f = open("shooting\_method")

x1 = []

y1 = []

n = int(f.readline())

for i in range(n):

x1.append(float(f.readline()))

for i in range(n):

y1.append(float(f.readline()))

exact\_y = [F(i) for i in x1]

plt.subplot(211)

plt.scatter(x1, y1, color="r")

plt.plot(x1, exact\_y, color='b')

plt.title('Shooting method')

f = open("finite\_diff\_method")

x2 = []

y2 = []

n = int(f.readline())

for i in range(n):

x2.append(float(f.readline()))

for i in range(n):

y2.append(float(f.readline()))

plt.subplot(212)

plt.scatter(x2, y2, color="r")

plt.plot(x2, exact\_y, color='b')

plt.title('Finite difference method')

plt.tight\_layout()

plt.show()

# **Выводы**

Существует более эффективный алгоритм подсчета числа точек на эллиптической кривой над конечным полем : алгоритм Шуфа. Он использует теорему Хасее и выполняется за время *O*(*log*8*q*).