**Московский Авиационный Институт**

**(национальный исследовательский университет)**

**Институт информационных технологий и прикладной математики**

**Кафедра вычислительной математики и программирования**

**Лабораторная работа №4 по курсу “Криптография”**

Студент: Жерлыгин М.А.

Преподаватель: Борисов А.В.

Группа: М8О-308Б-18

Дата:\_\_\_\_\_\_\_\_\_\_\_\_

Оценка:\_\_\_\_\_\_\_\_\_\_\_\_

Подпись:\_\_\_\_\_\_\_\_\_\_\_\_

**Задача:**Подобрать такую эллиптическую кривую над конечным простым полем порядка p, такую, порядок точки которой полным перебором находится за 10 минут на ПК. Упомянуть в отчёте какие алгоритмы и теоремы существуют для облегчения и ускорения решения задачи полного перебора.

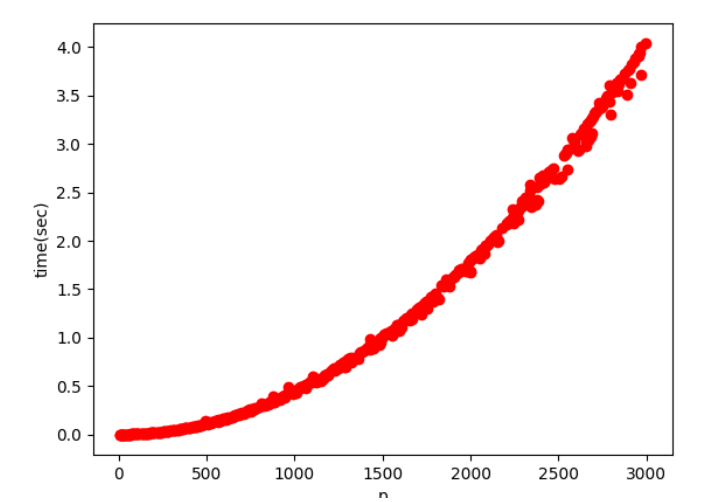
**Решение:**

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\_\_':  
 print("Start...")  
  
 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))

**Описание решения:**

Я взял эллиптическую кривую , выбрал случайным образом коэфициенты 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 график:



Из данного соотношения прикинула, что p должно быть около 37000.  
Поиск всех точек занимает много времени, это происходит из-за полного перебора (квадратичная сложность). 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.

**Результат работы:**

y^2 = x^3 + 12987 ∗ x + 32272 (mod 36997)  
Elliptic curve group order = 36953  
Order of point (17716, 12077): 36953  
Time: 624.449035168

**Вывод**

Существует более эффективный алгоритм подсчёта числа точек на эллиптической кривой над конечным полем - это алгоритм Шуфа. Он использует теорему Хасее и выполняется за время .