Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №6 по курсу**

**«Криптография»**

Студент: Почечура Артемий Андреевич

Группа: М80-306Б-20

Преподаватель: Борисов Август Валерьевич

Дата: 27.05.2023

Оценка:

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2023

**1. Этап вычисления**

Эллиптическая кривая строится согласно следующим условиям:

Коэффициенты *a* и *b* подбираются случайно в промежутке от (0, *p*). Коэффициент *p* – это простое число, по модулю которого берутся все значения. Чтобы программа работала чуть больше 10 минут (642 секунды), мною было выбрано значение *p* = 28871. До этого были проверены следующие значения:

* p = 13001 – 135 секунд;
* p = 25013 – 465 секунд;
* p = 33223 – 848 секунд;
* p = 30097 – 660 секунд;
* p = 29567 – 703 секунды.

Вычислительные мощности моего компьютера:

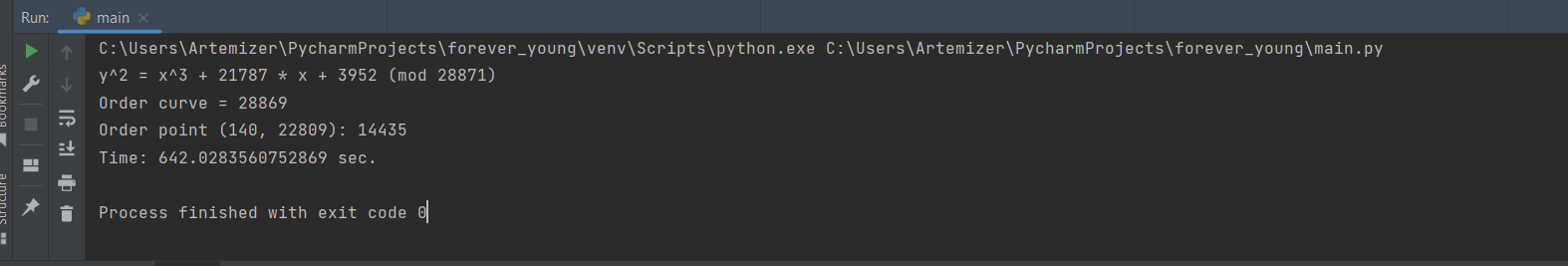
* Процессор – Intel Core i7 7700HQ с частотой 2800 МГц;
* Оперативная память – DDR4 8 ГБ с частотой 2400 МГц.

Работа программы длиться до тех пор, пока не будут перебраны все точки, координаты которых лежат в промежутке (0, *p*). Каждая точка проверялась на условие принадлежности эллиптической кривой. Очевидно, что сложность такого алгоритма . Порядок кривой определяется числом точек, которые ей принадлежат, а порядок точки в свою очередь определяется тем, сколько раз надо прибавить точку саму к себе, чтобы получить точку (0, 0).

Ниже приведён код программы на языке Python:

import time  
import random  
  
p = 28871  
a = random.randint(0, p)  
b = random.randint(0, p)  
while ((((4\*a)%p\*a)%p\*a)%p+((27\*b)%p\*b)%p)%p == 0:  
 a = random.randint(0, p)  
 b = random.randint(0, p)  
  
def extended\_euclidean\_algorithm(a, b):  
 *"""  
 Возвращает кортеж из трёх элементов (gcd, x, y), такой, что  
 a \* x + b \* y == gcd, где gcd - наибольший  
 общий делитель a и b.  
  
 В этой функции реализуется расширенный алгоритм  
 Евклида и в худшем случае она выполняется O(log 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):  
 *"""  
 Возвращает обратную величину  
 n по модулю p.  
  
 Эта функция возвращает такое целое число m, при котором  
 (n \* m) % p == 1.  
 """* gcd, x, y = extended\_euclidean\_algorithm(n, p)  
 assert (n \* x + p \* y) % p == gcd  
  
 if gcd != 1:  
 # Или n равно 0, или p не является простым.  
 raise ValueError(  
 '{} has no multiplicative inverse '  
 'modulo {}'.format(n, p))  
 else:  
 return x % p  
  
def new\_point(p1, p2, p):  
 if p1 == (0, 0):  
 return p2  
 elif p2 == (0, 0):  
 return p1  
 elif p1[0] == p2[0] and p1[1] != p2[1]:  
 return (0, 0)  
  
 if p1 == p2:  
 m = ((3 \* p1[0] \*\* 2 + (a % p)) \* inverse\_of(2 \* p1[1], p)) % p  
 else:  
 m = ((p1[1] - p2[1]) \* inverse\_of(p1[0] - p2[0], p)) % p  
 x = (m \*\* 2 - p1[0] - p2[0]) % p  
 y = (p1[1] + m \* (x - p1[0])) % p  
 return (x, -y % p)  
  
  
def order(point, p):  
 i = 1  
 tmp = point  
 while tmp != (0, 0):  
 tmp = new\_point(tmp, point, p)  
 i += 1  
 return i  
  
  
points = []  
start = time.time()  
for x in range(0, p):  
 for y in range(0, p):  
 if (y \*\* 2) % p == (x \*\* 3 + (a % p) \* x + (b % p)) % p:  
 points.append((x, y))  
print("y^2 = x^3 + {0} \* x + {1} (mod {2})".format(a % p, b % p, p))  
print("Order curve = {0}".format(len(points)))  
point = random.choice(points)  
print("Order point {0}: {1}".format(point, order(point, p)))  
print("Time: {} sec.".format(time.time() - start))

Результат работы программы:



**3. Вывод**

В процессе работы над данной лабораторной, я узнал, что такое эллиптические кривые и научился работать с ними. Эллиптические кривые можно эффективно применять в криптографии для генерации ключей, ведь для поиска параметра, на которое производится умножение, требуется решить задачу дискретного логарифмирования на эллиптической кривой. Пока что не существует алгоритма, который бы решал эту задачу за приемлемое время. Чтобы ускорить перебор всевозможных точек, можно применить алгоритм Шуфа, который использует теорему Хассе. Сложность такого алгоритма , где *q* – число элементов поля. Ещё можно использовать метод комплексного умножения, который позволяет эффективнее находить кривые, у которых задано число точек, но он требует выполнения некоторых условий.