**Московский авиационный институт**

**(Национальный исследовательский университет)**

Факультет прикладной математики и физики

Кафедра вычислительной математики и программирования

**Лабораторная работа № 3**

по курсу «Криптография»

Студент: Аксенов А. Е.

Группа: М80-306б-19

Преподаватель: Борисов А. В.

Оценка:

Москва, 2022

**Содержание**

1. Постановка задачи.

2. Общий метод и алгоритм решения.

3. Код программы.

3. Демонстрация работы.

4. Выводы.

1. **Постановка задачи**

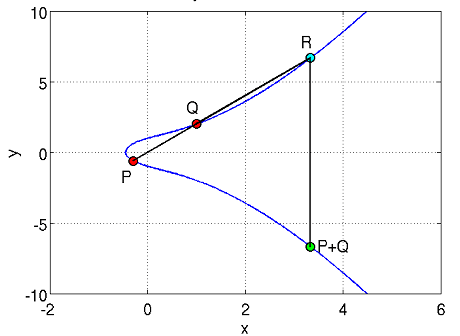
Подобрать такую эллиптическую кривую, порядок точки которой полным перебором находится за 10 минут на ПК. Упомянуть в отчёте результаты замеров работы программы, характеристики вычислителя. Также указать какие алгоритмы и/или теоремы существуют для облегчения и ускорения решения задачи полного перебора.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

1. **Общий метод и алгоритм решения**

Эллиптическая кривая задаётся уравнением:

, где *p* – порядок кривой.

Сложение двух точек **P** и **Q** на эллиптической кривой можно представить следующим образом (на примере кривой ):



Как видно из рисунка, для сложения точек **P** и **Q**, необходимо провести между ними прямую линию, которая обязательно пересечет кривую в какой-либо третьей точке **R**. Отразим точку **R** относительно горизонтальной оси координат и получим искомую точку **P**+**Q**.

Скалярное произведение точки **P** на число **n** можно определить как n-кратную сумму числа **P** с самим собой.

Порядком точки **P** называется такое натуральное число **n**, что **nP** = **P**.

Основная задача заключается в том, чтобы подобрать числа a, b, p и координаты точки x, y таким образом, чтобы поиск порядка точки с данными координатами путем перебора выполнялся за 10 минут или более. В качестве таких чисел я взял:

a = 2

b = 62730594

p = 96120229

x = 53479085

y = 38714391

Для реализации поиска порядка, я воспользовался питоном, чтобы написать класс точки, в котором определил метод сложения и метод перебора чисел.

Вычисления производились на ноутбуке со следующими характеристиками: *Процессор:* Intel® Core™ i5-8250U CPU 1.60GHz 3.40GHz,

*Память:* 16ГБ,

*ПО:* ОС Windows 10 64-разрядная, WSL2 Ubuntu 20.04 LTS.

1. **Код программы**

from time import gmtime, strftime, time

# y\*\*2 = x\*\*3 + a\*x + b

class Point:

x = 0

y = 0

p = 0

a = 0

def \_\_init\_\_(self, x, y, p, a):

self.x = x

self.y = y

self.p = p

self.a = a

def \_\_repr\_\_(self):

return "Point()"

def \_\_str\_\_(self):

return "(" + str(self.x) + ", " + str(self.y) + ")"

def \_\_eq\_\_(self, other):

return self.x == other.x and self.y == other.y

def \_\_ne\_\_(self, other):

return not self == other

def copy(self):

return Point(self.x, self.y, self.p, self.a)

def is\_zero(self):

return self.x == 0 and self.y == 0

def extended\_euclidean\_algorithm(self, 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\_mod(self, k):

if k == 0:

raise ZeroDivisionError('division by zero')

if k < 0:

return self.p - self.inverse\_mod(-k)

s, old\_s = 0, 1

t, old\_t = 1, 0

r, old\_r = self.p, k

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

gcd, x, y = old\_r, old\_s, old\_t

assert gcd == 1

assert (k \* x) % self.p == 1

return x % self.p

def add(self, other):

if self.is\_zero():

return other

if other.is\_zero():

return self

if self.x == other.x and self.y != other.y:

return Point(0, 0, self.p, self.a)

if self == other:

m = ((3 \* self.x \* self.x + self.a) \* self.inverse\_mod(2 \* other.y)) % self.p

else:

m = ((self.y - other.y) \* self.inverse\_mod(self.x - other.x)) % self.p

x = (m \* m - self.x - other.x) % self.p

y = (self.y + m \* (x - self.x)) % self.p

return Point(x, -y % self.p, self.p, self.a)

# удвоение

def x2(self):

if self.is\_zero():

return self.copy()

try:

L = ((3 \* self.x \* self.x + self.a) / (2 \* self.y)) % self.p

except ZeroDivisionError:

return Point(self.p, self.p, self.p, self.a)

x = (L \* L - 2 \* self.x) % self.p

y = (L \* (self.x - x) - self.y) % self.p

return Point(x, y, self.p, self.a)

# алгоритм поиска порядка точки

# складываем точку с собой k раз, пока не получим исходную точку

def find(self):

t1 = time()

k = 1

sum = self.copy()

sum = sum.add(self)

while sum != self:

k = k + 1

sum = sum.add(self)

t2 = time()

print("find: ")

print(str(sum))

print("k: " + str(k))

print("Time = %s" % strftime('%H:%M:%S', gmtime(t2 - t1)))

a = 2

b = 62730594

p = 96120229

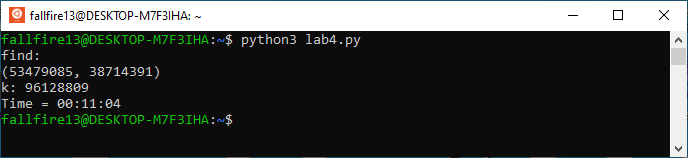
x = 53479085

y = 38714391

p1 = Point(x, y, p, a)

p1.find()

1. **Демонстрация работы**



1. **Выводы**

Как мы видим, полный перебор порой неэффективен по времени, однако помимо него существуют также более эффективные способы нахождения порядка точки. Приведу некоторые из них:

1) Алгоритм ***Шенкса*** (***baby-step***, ***giant-step)***: сначала мы вычисляем несколько точек с небольшим шагом и сохраняем их в хеш-таблице. Затем делаем великанские шаги и сравниваем новые точки с точками в хеш-таблице. Найдя соответствие, мы можем вычислить дискретный алгоритм простой перестановкой членов.

2) Алгоритм ***Полларда***: задаем псевдослучайную последовательность пар (a, b), генерируем по ней последовательность точек aP + bQ. Данные последовательности будут являться циклическими. Как только мы обнаружим цикл, то есть найдем такие 2 пары (a, b) и (A, B), что aP+bQ = AP+BQ, нам останется воспользоваться формулой n=a-Ab-B-1mod p

3) Алгоритм ***Шора***: разработан для решения данной задачи на квантовых компьютерах.