Отчёт по лабораторной работе №5 по курсу «Криптография»

Выполнил *Попов Николай,* группа *М8О-308Б-21*

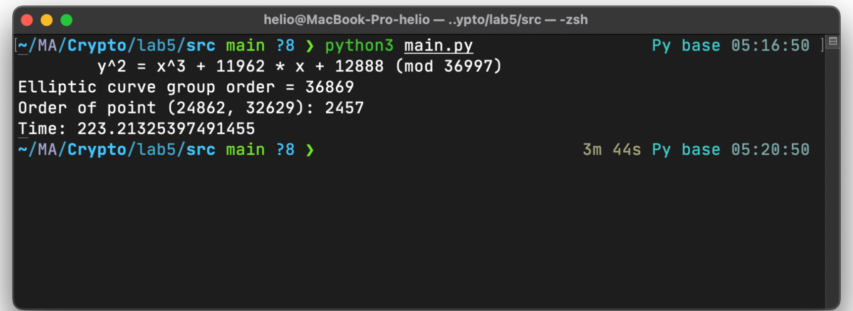
Задание

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 выбрал эллиптическую кривую $y^2 = x^3 + ax + b$ и случайным образом зафиксировал коэффициенты a и b. С помощью решета Эратосфена я сформировал массив простых чисел до 3000 и измерил, сколько времени занимает для различных значений p поиск всех точек на кривой, а также нахождение порядка случайно выбранной точки. Результаты эксперимента представлены ниже.

Результат:



Эллиптическая кривая: $y^2 = x^3 + 11962x + 12888 \pmod{36997}$

Получена за 223 секунды

Вывод:

Выполняя эту работу, я познакомился с криптографией на эллиптических кривых. Полным перебором я подобрал некоторую эллиптическую кривую, также я узнал, что существует более эффективный алгоритм подсчёта числа точек на эллиптической кривой над конечным полем: алгоритм Шуфа. Он использует теорему Хасее и выполняется за время $O(log^8 q)$.

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))
```