МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

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 дисциплине «Криптография»

Выполнила студент группы М80-307Б-20:
Сафонникова А. Р.

Принял:
Борисов А. В.

Москва 2023

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.

РЕШЕНИЕ

Каноническая форма эллиптической кривой: y^2 = x^3 + ax +b.

Коэффициенты a и b выберем рандомно.

Напишем следующую программу:

```
import random
import time
import matplotlib.pyplot as plt
import numpy as np

def generate_random_points(p, A, B):
    def elliptic_curve(x, y, p):
        return (y ** 2) % p == (x ** 3 + (A % p) * x + (B % p)) % p

    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 % b, old_t % a

def inverse(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

def add_points(p1, p2, p, A):
    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((2 * y1) % p, p)) % p
    else:
        m = ((y1 - y2) * inverse((x1 - x2) % p, p)) % p

    x3 = ((m ** 2) - x1 - x2) % p
    y3 = (y1 + m * (x3 - x1)) % p

    return [x3, -y3 % p]

def point_order(point, p, A):
    i = 1
    new_point = add_points(point, point, p, A)
    while new_point != (0, 0):
        new_point = add_points(new_point, point, p, A)
        i += 1

    return i

def sieve(n):
    primes = 2 * [False] + (n - 1) * [True]
    for i in range(2, int(n ** 0.5 + 1.5)):
        if primes[i]:
            primes[i*i::i] = [False] * len(primes[i*i::i])
    return [prime for prime, checked in enumerate(primes) if checked]

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

def run(sugg):
    primes = sieve(sugg)
    p = primes[-1]

    start = time.time()

    points = generate_random_points(p, A, B)

    points_num = len(points)

    print_curve_equation(p, A, B)
    print("Elliptic curve order = {0}".format(points_num))

    point = random.choice(points)
```

```
    print("Point order {0}: {1}".format(point, point_order(point, p,
A)))
    print("Time: {0}".format(time.time() - start))


if __name__ == '__main__':
    A = random.randint(1000000000, 10000000000)
    B = random.randint(1000000000, 10000000000)

    suggest = [23000]

    for i in suggest:
        print('P suggestion = {0}'.format(i))
        run(i)
        print()
```

Этот код реализует вычисление порядка эллиптической кривой над конечным полем. Рассмотрим каждую из функций:

- В *generate_random_points* мы генерируем случайные точки на кривой.

- Функция *extended_euclidean_algorithm* реализует расширенный алгоритм Евклида для вычисления обратного элемента n по модулю p.

- Функция *inverse* использует расширенный алгоритм Евклида для нахождения обратного элемента n по модулю p.

- В функции *add_points* мы реализуем операцию сложения двух точек на эллиптической кривой.

- Функция *point_order* вычисляет порядок заданной точки на эллиптической кривой.

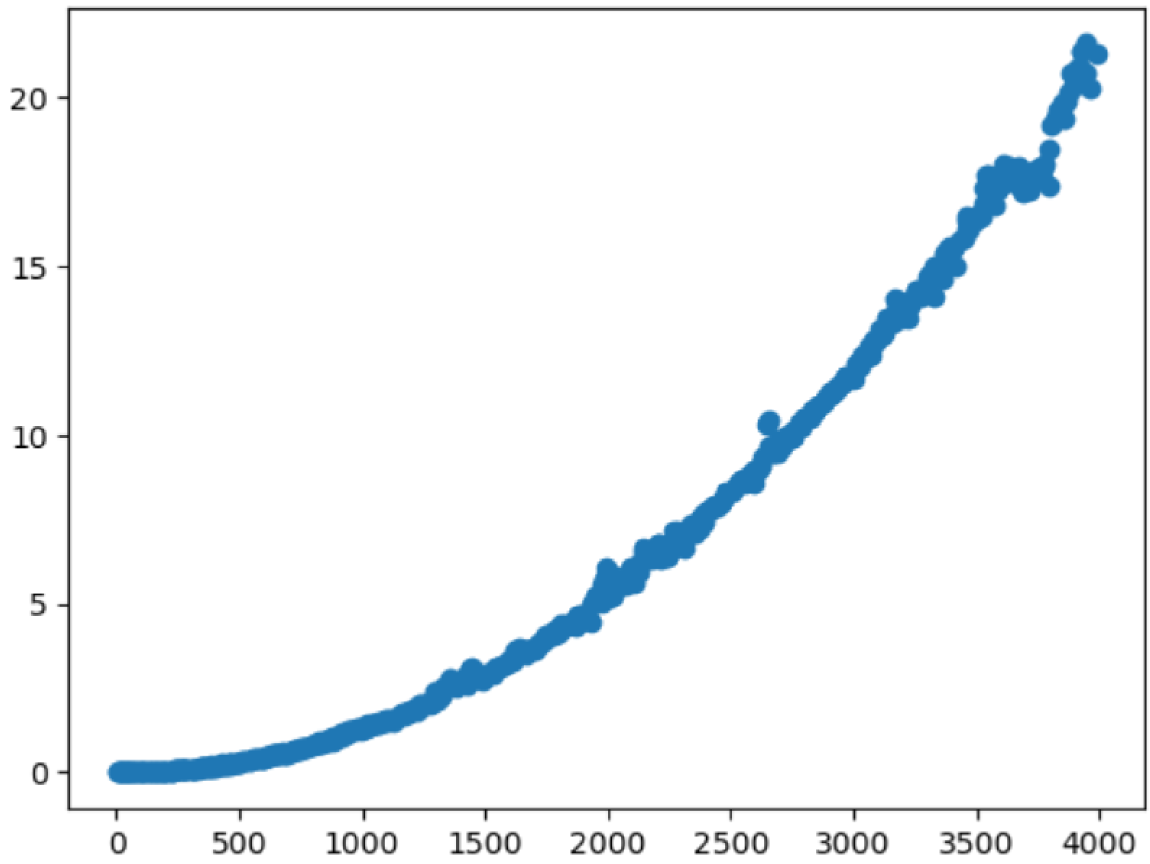
- Функция *sieve* выполняет алгоритм "Решето Эратосфена" для поиска всех простых чисел до n.

- Функция *print_curve_equation* просто выводит уравнение эллиптической кривой.

- В функции *run* мы выбираем p из списка простых чисел primes и вызываем все необходимые функции для вычисления порядка и времени выполнения.

В основной части программы, мы генерируем случайные значения A и B, выбираем значение p из списка suggest, и вызываем функцию run для выполнения вычислений.

Для подбора подходящего p построим график зависимости затраченного времени от величины p:



Значит, p мы будем брать в промежутке от 20000 до 25000. А теперь попробуем пару значений и найдём подходящее:

```
P suggestion = 15000
y^2 = x^3 + 13708 * x + 655 (mod 14983)
Elliptic curve order = 14871
Point order (7782, 3320): 337
Time: 383.09304785728455
```

```
P suggestion = 25000
y^2 = x^3 + 12354 * x + 21093 (mod 24989)
Elliptic curve order = 24971
Point order (19338, 13175): 2080
Time: 1045.793470621109
```

```
P suggestion = 20000
y^2 = x^3 + 17549 * x + 15503 (mod 19997)
Elliptic curve order = 19926
Point order (9111, 12970): 19926
Time: 523.350818157196
```

```
P suggestion = 20500
y^2 = x^3 + 16618 * x + 1672 (mod 10483)
Elliptic curve order = 10557
Point order (1082, 14133): 540
Time: 520.9275722503662
```

```
P suggestion = 22000
y^2 = x^3 + 1902 * x + 6261 (mod 21997)
Elliptic curve order = 22103
Point order (9888, 18392): 3683
Time: 533.2623534202576
```

```
P suggestion = 23000
y^2 = x^3 + 6834 * x + 9024 (mod 22993)
Elliptic curve order = 23088
Point order (22420, 15202): 23088
Time: 633.4173350334167
```

Ура! Значит, значение p = 23000, а результаты показывают, что кривая заданного вида имеет порядок 23088, а выбранная случайная точка (22420, 15202) имеет порядок 23088.

ВЫВОД

Существует несколько алгоритмов, способных ускорить решение данной задачи. Так, есть алгоритм Шуфа, способный решить задачу вычисления порядка кривой за сложность , что намного лучше наивного решения.

Также есть возможность ускорить решение задачи поиска порядка точки. Для этого можно применить алгоритмы «baby-step, giant-step» и ρ-алгоритм Полларда оба алгоритма решают задачу дискретного логарифмирования: найти для двух заданных точек и целое число x, удовлетворяющее уравнению.

Также, полезным фактом будет то, что порядок точки всегда является делителем порядка кривой. Данное свойство происходит из теоремы Лагранжа и на его основе можно придумать оптимизацию. Для определения порядка точки нам достаточно определились порядок кривой, факторизовать его каким-нибудь быстрым алгоритмом и затем перебирать делители в качестве потенциальных ответов, существенно сократив область поиска.

Таким образом, решение задачи полного перебора для эллиптических кривых требует тщательного выбора параметров и может быть улучшено с помощью более продвинутых методов, учитывающих свойства кривых и поля.