Отчёт по лабораторной работе 4

Студент: Кочкожаров Иван Вячеславович
Группа: М8О-308Б-22

1 Цель работы

Подобрать такую эллиптическую кривую, порядок точки которой полным перебором находится за 10 минут на ПК. Исследовать алгоритмы и теоремы, существующие для облегчения и ускорения решения задачи полного перебора.

2 Теоретическая часть

2.1 Эллиптические кривые над конечным полем

Эллиптическая кривая над конечным полем $\mathbb{F}_p$ (где p — простое число) определяется уравнением:

$$y^2 \equiv x^3 + ax + b \pmod{p} \tag{1}$$

При условии, что $4a^3 + 27b^2 \not\equiv 0 \pmod{p}$ (условие несингулярности).

Точки эллиптической кривой образуют абелеву группу с операцией сложения, определенной геометрически. Порядок точки P — это наименьшее положительное целое число n такое, что $nP = \mathcal{O}$ (бесконечно удаленная точка).

2.2 Оценка производительности

Для оценки среднего времени выполнения одной операции сложения точек был проведен эксперимент с кривой над полем $\mathbb{F}_p$ при $p = 50\,000\,017$:

```
p=50000017, a=6470481, b=8975775, P=(12195349,21713483), order
   =49992337, time=100.81s
Avg iter time = 2.0164165265300723e-06
```

Среднее время одной операции сложения составило примерно $2.02 \cdot 10^{-6}$ с.

2.3 Оценка времени вычисления порядка точки

Если каждая операция сложения занимает примерно $2 \cdot 10^{-6}$ с (2 μs), то за 10 минут (600 с) вы сможете сделать порядка

$$\frac{600 \text{ с}}{2 \cdot 10^{-6} \text{ с/операцию}} = 3 \cdot 10^8 \quad \text{скалярных сложений.}$$

Поскольку порядок точки на «случайной» кривой над $\mathbb{F}_p$ примерно равен p (в пределах $\pm\sqrt{p}$) по теореме Хассе, чтобы перебор kP для $k = 1 \dots p$ занял около 600 с, достаточно взять

$$p \approx 3 \cdot 10^8.$$

Например, одно из ближайших простых чисел порядка $6 \cdot 10^8$ —

$$\boxed{p = 300\,000\,007.}$$

Его используем для начального приближения кривой.

3 Практическая часть

3.1 Реализация алгоритмов

Для решения задачи были реализованы следующие компоненты:

1. Класс `EllipticCurve` для представления эллиптической кривой
2. Класс `Point` для представления точек на кривой
3. Функции для операций над точками: `add` и `mul`
4. Функция `brute_order` для вычисления порядка точки методом полного перебора
5. Функции для генерации случайных кривых и точек

3.2 Подбор параметров кривой

Для поиска подходящей кривой был реализован алгоритм, который:

1. Генерирует случайную эллиптическую кривую над полем $\mathbb{F}_p$
2. Генерирует случайную точку на этой кривой
3. Вычисляет порядок точки методом полного перебора
4. Проверяет, что время вычисления находится в заданном диапазоне (9-11 минут)

Были проведены эксперименты с различными значениями p:

- $p = 300\,000\,007$ (результаты оказались слишком быстрыми, около 440 с)
- $p = 426\,000\,017$ (найдена кривая с временем вычисления 658.79 с)
- $p = 387\,272\,759$ (найдена кривая с временем вычисления 621.83 с)

4 Результаты

В результате экспериментов была найдена оптимальная эллиптическая кривая:

$$y^2 = x^3 + 124\,860\,295x + 2\,186\,117 \pmod{387\,272\,759} \tag{2}$$

$$P = (223\,741\,075, 161\,701\,677) \tag{3}$$

$$\text{Порядок точки } P = 387\,264\,450 \tag{4}$$

$$\text{Время вычисления} = 621.83 \text{ с} \tag{5}$$

Это значение находится в пределах заданного диапазона (9-11 минут) и подтверждает теоретические оценки.

5 Алгоритмы для ускорения вычисления порядка точки

Существуют более эффективные алгоритмы для вычисления порядка точки на эллиптической кривой:

5.1 Алгоритм Шенкса (Baby-step Giant-step)

Алгоритм Шенкса позволяет найти порядок точки за время $O(\sqrt{n})$, где n — порядок группы. Основная идея:

1. Выбрать параметр $m \approx \sqrt{n}$
2. Вычислить и сохранить точки jP для $j = 0, 1, \ldots, m-1$
3. Вычислить точки $P_0 - imP$ для $i = 0, 1, \ldots, m$, где P_0 — некоторая известная точка
4. Найти совпадение между двумя наборами точек

5.2 Алгоритм Полларда ρ

Алгоритм Полларда использует псевдослучайную функцию для поиска цикла в последовательности точек. Время работы также $O(\sqrt{n})$, но требует меньше памяти, чем алгоритм Шенкса.

5.3 Теорема Хассе

Теорема Хассе утверждает, что порядок эллиптической кривой E над полем $\mathbb{F}_p$ находится в диапазоне:

$$p + 1 - 2\sqrt{p} \leq \#E(\mathbb{F}_p) \leq p + 1 + 2\sqrt{p} \tag{6}$$

Это позволяет существенно сузить область поиска порядка кривой.

5.4 Алгоритм Шуфа

Алгоритм Шуфа позволяет вычислить точное значение порядка эллиптической кривой за полиномиальное время. Он основан на вычислении порядка кривой по модулю малых простых чисел и последующем применении Китайской теоремы об остатках.

6 Характеристики вычислителя

- Процессор: AMD Ryzen 5 5600H (8 ядер, 16 потоков)
- Оперативная память: 16 ГБ DDR4 3200 МГц
- Операционная система: Arch Linux (ядро 6.14.6-arch1-1)
- Python: версия 3.13.3

7 Выводы

1. Была подобрана эллиптическая кривая, для которой вычисление порядка точки методом полного перебора занимает около 10 минут (621.83 с).
2. Экспериментально подтверждены теоретические оценки времени вычисления.
3. Рассмотрены более эффективные алгоритмы для вычисления порядка точки, которые могут значительно ускорить процесс по сравнению с полным перебором.
4. Полученные результаты демонстрируют важность использования оптимизированных алгоритмов при работе с эллиптическими кривыми в криптографических приложениях.

8 Приложение: исходный код

https://github.com/kochkozharov/cryptography-labs/tree/master/lab4

8.1 ec.py - Реализация эллиптических кривых

```python
import random
import signal
import time


class EllipticCurve:
    def __init__(self, a, b, p):
        self.a = a
        self.b = b
        self.p = p
        if (4 * a**3 + 27 * b**2) % p == 0:
            raise ValueError("Singular curve")

class Point:
    def __init__(self, x=None, y=None, curve=None):
        self.x = x
        self.y = y
        self.curve = curve

    def is_infinity(self):
        return self.x is None and self.y is None

    @staticmethod
    def infinity(curve):
        return Point(None, None, curve)

def add(P, Q):
    curve = P.curve
    p = curve.p
    if P.is_infinity(): return Q
    if Q.is_infinity(): return P
    if P.x == Q.x and (P.y + Q.y) % p == 0:
        return Point.infinity(curve)
```

```
    if P.x == Q.x:
        lam = (3 * P.x * P.x + curve.a) * pow(2 * P.y, -1, p) % p
    else:
        lam = ((Q.y - P.y) * pow(Q.x - P.x, -1, p)) % p
    x_r = (lam * lam - P.x - Q.x) % p
    y_r = (lam * (P.x - x_r) - P.y) % p
    return Point(x_r, y_r, curve)

def mul(P, n):
    R = Point.infinity(P.curve)
    Q = P
    while n > 0:
        if n & 1:
            R = add(R, Q)
        Q = add(Q, Q)
        n >>= 1
    return R


def get_random_xy(E):
    while True:
        x = random.randrange(E.p)
        rhs = (x**3 + E.a*x + E.b) % E.p
        y = None
        for yy in range(E.p):
            if yy*yy % E.p == rhs:
                y = yy
                break
        if y:
            break
        else:
            continue
    return x, y

def get_random_curve(p):
    while True:
        a = random.randrange(p)
        b = random.randrange(p)
        try:
            E = EllipticCurve(a, b, p)
            break
        except ValueError:
            continue
    return E

class TimeoutError(Exception):
    pass

def _timeout_handler(signum, frame):
    raise TimeoutError


def brute_order(P, max_seconds=660):
```

```
    signal.signal(signal.SIGALRM, _timeout_handler)
    signal.alarm(max_seconds)
    start = time.perf_counter()
    try:
        R = Point.infinity(P.curve)
        n = 1
        while True:
            R = add(R, P)
            if R.is_infinity():
                break
            n += 1
        elapsed = time.perf_counter() - start
        signal.alarm(0)
        return n, elapsed
    except TimeoutError:
        elapsed = time.perf_counter() - start
        signal.alarm(0)
        return None, elapsed
```

8.2 main.py - Поиск подходящей кривой

```
from ec import *


def find_curve(p, min_seconds=540, max_seconds=660):
    while True:
        E = get_random_curve(p)
        x, y = get_random_xy(E)
        P = Point(x, y, E)
        order, elapsed = brute_order(P, max_seconds)
        print(f"p={p}, a={E.a}, b={E.b}, P=({x},{y}), order={order},
           time={elapsed:.2f}s")
        if order is None:
            print(f"Timeout after {max_seconds}s")
            continue
        if elapsed < min_seconds:
            print(f"Too fast: only {elapsed:.2f}s")
            continue
        return E, P, order, elapsed

if __name__ == '__main__':
    p = 387272759
    curve, point, order, elapsed = find_curve(p)
    print("\nSelected curve and point:")
    print(f"y^2 = x^3 + {curve.a}x + {curve.b} (mod {curve.p})")
    print(f"Point P = ({point.x}, {point.y}), order = {order}, computed
        in {elapsed:.2f}s")
```

8.3 estimation.py - Измерение среднего времени операции

```
from ec import *
```

```
import time

if __name__ == '__main__':
    p = 50000017
    E = get_random_curve(p)
    x, y  = get_random_xy(E)
    P = Point(x, y, E)
    time_sum=0
    R = Point.infinity(P.curve)
    n = 1
    while True:
        start = time.perf_counter()
        R = add(R, P)
        if R.is_infinity():
            break
        n += 1
        elapsed = time.perf_counter() - start
        time_sum+=elapsed
    print(f"p={p}, a={E.a}, b={E.b}, P=({x},{y}), order={n}, time={
       time_sum:.2f}s")
    print(f"Avg iter time = {time_sum / (n-1)}")
```