Отчёт по лабораторной работе №6 по курсу «Криптография»

Выполнил Попов Матвей, группа М8О-308Б-20

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.

Ход работы

Для данной лабораторной работы было решено использовать язык Java, так как программы на нём выполняются значительно быстрее, чем на Python, но медленнее, чем на компилируемых языках, таких как C и C++, а значит подбор будет наиболее справедливым.

Характеристики вычислителя:

- процессор: AMD Ryzen 7 4800U (8 ядер, 16 потоков, макс. частота 4.2 ГГц)
- оперативная память: 16 Гб DDR4

Во время вычислений ПК работал в сбалансированном режиме производительности

Результат работы программы

Зелёным цветом и курсивом выделены введённые данные

```
Enter a:
2562356
Enter b:
8764876
Enter p:
503
Enter time in seconds:
600

Iteration 1
y^2 = x^3 + 2837*x + 1420 % 3511
Curve order: 3479
Point (1589, 2624) order: 221
Elapsed time: 1 seconds

Iteration 2
y^2 = x^3 + 3897*x + 5245 % 9511
Curve order: 9673
Point (8191, 7377) order: 1352
Elapsed time: 9 seconds
```

```
Iteration 3
y^2 = x^3 + 7010*x + 6335 % 18517
Curve order: 18759
Point (3453, 6089) order: 10667
Elapsed time: 34 seconds

Iteration 4
y^2 = x^3 + 29445*x + 6497 % 30517
Curve order: 30553
Point (26185, 23438) order: 14433
Elapsed time: 105 seconds

Iteration 5
y^2 = x^3 + 13068*x + 24460 % 45523
Curve order: 45636
Point (24766, 32964) order: 35061
Elapsed time: 246 seconds

Iteration 6
y^2 = x^3 + 21276*x + 61677 % 63527
Curve order: 63542
Point (35604, 23609) order: 25896
Elapsed time: 454 seconds

Iteration 7
y^2 = x^3 + 26366*x + 57977 % 84533
Curve order: 84260
Point (36291, 26720) order: 35970
Elapsed time: 802 seconds

Resulting elliptic curve: y^2 = x^3 + 26366*x + 57977 % 84533
```

Таким образом, ушло 7 итераций, чтобы найти нужную эллиптическую кривую. Каждая следующая итерация занимала примерно в 2-3 раза больше времени, чем предыдущая.

Результат: $y^2 = x^3 + 26366x + 57977$

Вывод

Проделав лабораторную работу, я за 802 секунды полным перебором подобрал эллиптическую кривую. Возможно, можно было бы облегчить решения, например для проверки числа на простоту использовать решето Эратосфена, а для облегчения перебора применить теоремы Хассе и Шуфа.

Листинг программы

lab06/src/Point.java

```
public record Point(long x, long y) {

    @Override
    public boolean equals(Object obj) {
        if (this == obj) {
            return true;
        }
        if (obj == null || getClass() != obj.getClass()) {
            return false;
        }
        Point other = (Point) obj;
```

```
        return this.x == other.x && this.y == other.y;
    }

    @Override
    public String toString() {
        return "(" + this.x + ", " + this.y + ")";
    }
}
```

lab06/src/EllipticCurve.java

```
import java.util.ArrayList;
import java.util.Random;

public class EllipticCurve {

    private final long A;
    private final long B;
    private long p;

    private long ap;
    private long bp;

    private final Random random = new Random(2904);

    public long getP() {
        return p;
    }

    public void setP(long p) {
        this.p = p;
    }

    public EllipticCurve(long A, long B, long P) {
        this.A = A;
        this.B = B;
        this.p = P;

        this.ap = this.A % this.p;
        this.bp = this.B % this.p;
    }

    public boolean isEllipticCurve(long x, long y) {
        return (Math.pow(y, 2)) % this.p == (Math.pow(x, 3) + this.ap * x + this.bp) %
this.p;
    }

    @Override
    public String toString() {
        return "y^2 = x^3 + " + this.ap + "*x + " + this.bp + " % " + this.p;
    }

    private static long[] extendedEuclideanAlgorithm(long a, long b) {
        long s = 0, t = 1, r = b;
        long oldS = 1, oldT = 0, oldR = a;

        while (r != 0) {
            long quotient = oldR / r;
            oldR = r;
            r = oldR - quotient * r;
            oldS = s;
            oldT = t;
            t = oldT - quotient * t;
        }

        long[] res = new long[3];
        res[0] = oldR;
        res[1] = oldS;
        res[2] = oldT;
```

```
    return res;
}

private long inverseOf(long n) {
    long[] res = extendedEuclideanAlgorithm(n, this.p);
    long gcd = res[0], x = res[1], y = res[2];

    if ((n * x + p * y) % p == gcd) {
        return -1;
    }
    if (gcd != 1) {
        return -1;
    } else {
        return x % this.p;
    }
}

private Point addPoints(Point p1, Point p2) {
    long s;

    if (p1.equals(new Point(0, 0))) {
        return p2;
    } else if (p2.equals(new Point(0, 0))) {
        return p1;
    } else if (p1.x() == p2.x() && p1.y() != p2.y()) {
        return new Point(0, 0);
    }

    if (p1.equals(p2)) {
        s = ((3 * p1.x() * p1.x() + this.ap)) * inverseOf(2 * p1.y()) % this.p;
    } else {
        s = ((p1.y() - p2.y()) * inverseOf(p1.x() - p2.x())) % this.p;
    }

    long x = (long) ((Math.pow(s, 2) - 2 * p1.x()) % this.p);
    long y = (p1.y() + s * (x - p1.x())) % this.p;

    return new Point(x, -y % this.p);
}

private long orderPoint(Point point) {
    long i = 0;
    Point check = addPoints(point, point);

    while (!check.equals(new Point(0, 0))) {
        check = addPoints(check, point);
        i++;
    }
    return i;
}

public long step() {
    ap = A % p;
    bp = B % p;
    System.out.println(this);

    ArrayList<Point> points = new ArrayList<>();
    long startTime = System.nanoTime();

    for (long x = 0; x != this.p; x++) {
        for (long y = 0; y != this.p; y++) {
            if (isEllipticCurve(x, y)) {
                points.add(new Point(x, y));
            }
        }
    }

    System.out.println("Curve order: " + points.size());
```

```
        int randomIndex = random.nextInt(points.size());
        Point randomPoint = points.get(randomIndex);
        System.out.println("Point " + randomPoint + " order: " + orderPoint(randomPoint));

        long elapsedTime = System.nanoTime() - startTime;
        elapsedTime = elapsedTime / 1000000000;
        System.out.println("Elapsed time: " + elapsedTime + " seconds");
        System.out.println();
        return elapsedTime;
    }

    private boolean isPrimeNumber(long number) {
        boolean isPrime = true;

        for (long i = 2; i != number; i++) {
            if (number % i == 0) {
                isPrime = false;
                break;
            }
        }
        return isPrime;
    }

    public long getNextPrimeNumber(long start) {
        while (!isPrimeNumber(start)) {
            start++;
        }
        return start;
    }
}
```

lab06/src/Main.java

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        long a, b, p, timeToCalculate;

        Scanner in = new Scanner(System.in);
        System.out.println("Enter a:");
        a = in.nextLong();
        System.out.println("Enter b:");
        b = in.nextLong();
        System.out.println("Enter p:");
        p = in.nextLong();
        System.out.println("Enter time in seconds:");
        timeToCalculate = in.nextLong();
        System.out.println();

        EllipticCurve ec = new EllipticCurve(a, b, p);

        long timePassed = 0;
        long iter = 1;

        while (timePassed < timeToCalculate) {
            System.out.println("Iteration " + iter);
            ec.setP(ec.getNextPrimeNumber(ec.getP() + iter * 3000));
            timePassed = ec.step();
            iter++;
        }

        System.out.println("Resulting elliptic curve: " + ec);
    }
}
```