ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Факультет программной инженерии и компьютерной техники

ЛАБОРАТОРНАЯ РАБОТА № 2.6

ПО ДИСЦИПЛИНЕ «ИНФОРМАЦИОННАЯ БЕЗОПАСНОСТЬ»

Расшифрование криптограммы на основе эллиптических кривых

Выполнил: Давыдов Иван Денисович

Группа: Р3400

Вариант 5

Санкт-Петербург
2020/2021

Цель работы

Расшифровать текст, используя алфавит, приведенный в задании к лабораторной (используется кривая $E_{751}(-1, 1)$: $y^2 = x^3 - 1x + 1 \pmod{751}$).

Задание

№ варианта	Секретный ключ nb	Шифртекст
5	41	{(283, 493), (314, 127)}; {(425, 663), (561, 140)}; {(568, 355), (75, 433)}; {(440, 539), (602, 627)}; {(188, 93), (395, 414)}; {(179, 275), (25, 604)}; {(72, 254), (47, 349)}; {(72, 254), (417, 137)}; {(188, 93), (298, 225)}; {(56, 419), (79, 111)}

Листинг

Point.java

```
package core;

public class Point {
    private int x;
    private int y;


    Point(int x, int y){
        this.x = x;
        this.y = y;
    }

    int getX() {
        return x;
    }

    int getY() {
        return y;
    }

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

        Point p = ((Point) obj);
        return this.x == p.getX() && this.y == p.getY();
    }

    @Override
    public int hashCode() {

        return new Integer(x+y).hashCode();
    }

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

EllepticCurve.java

```java
package core;

class EllepticCurve {
    private int a, b, p;

    EllepticCurve(int a, int b, int p){
        this.a = a;
        this.b = b;
        this.p = p;
    }

    // Умножаем по быстрому алгоритму удвоения-сложения
    // 41 == 0b101001 => 41P = 2^5P + 2^3P + 2^0P
    Point mul(Point p, int n){
        int tmp = 1, power = -1;
        Point resP = null, tmpP = new Point(p.getX(), p.getY());;

        while (tmp <= n) {
            power++;
            tmp = tmp << 1;
        }

        tmp = n;
        boolean first = true;
        for (int i = 0; i <= power; i++){
            if((tmp & 1) == 1) {
                if(first) {
                    resP = new Point(tmpP.getX(), tmpP.getY());
                    first = false;
                }
                else
                    resP = this.sum(resP, tmpP);

            }

            tmpP = this.sum(tmpP, tmpP);
            tmp = tmp >>> 1;
        }

        return resP;
    }

    // Метод расчета суммы двух точек на эллиптической кривой
    Point sum(Point p1, Point p2) {

        /*
            x3 = λ^2 - x1 - x2(mod p)
            y3 = λ(x1 - x3) - y1(mod p)
        */
        int lambda = calcLambda(p1, p2);
        int tmp = lambda * lambda - p1.getX() - p2.getX();
        int newX = tmp >= 0 ? tmp % this.p : this.p + (tmp % this.p);

        tmp = lambda * (p1.getX() -newX) - p1.getY();
        int newY = tmp >= 0 ? tmp % this.p : this.p + (tmp % this.p) ;

        Point res = new Point(newX, newY);


        return new Point(newX, newY);
    }
```

```
    // метод расчета наклона прямой, проходящей через две точки
    private int calcLambda(Point p1, Point p2 ) {
        int numerator, denominator;

        // p1 == p2: λ = (3x1^2 + a) / 2y1
        // p1 != p2: λ = (y2 - y1) / (x2 - x1)
        if (p1.equals(p2)) {
            numerator = 3 * p1.getX() * p1.getX() + this.a;
            denominator = 2 * p1.getY();
        }
        else {
            numerator = p2.getY() - p1.getY();
            denominator = p2.getX() - p1.getX();
        }

        // a / b = a * b^(-1); Ищем обратную величину по модулю
        denominator = this.invMod(denominator);

        // если вычисленное значение получится отрицательным, приводим к положительному
        return (numerator * denominator < 0) ?
                this.p + (numerator * denominator) % this.p :
                (numerator * denominator) % this.p;
    }


    //Возвращает обратную величину n по модулю p
    //такое целое число m, при котором (n*m)%p == 1
    //Применяется расширенный алгоритм Евклида
    private int invMod(int n){
        int s = 0, oldS = 1;
        int r = this.p, oldR = n;

        while (r != 0){
            int quotient = (int)Math.floorDiv(oldR , r);
            int tmp = r;
            r = oldR - (quotient * r);
            oldR = tmp;

            tmp = s;
            s = oldS - quotient * s;
            oldS = tmp;

        }

        //учитываем отрицательный результат
        return oldS + (oldS < 0 ? this.p : 0);
    }
}
```

Main.java

```
package core;

import java.util.HashMap;
import java.util.List;

public class Main {

    public static void main(String[] args) {
        EllepticCurve curve;
        int nB;
        List<Point[]> cipherText;

        // считываем алфавит и вариант из файлов
        HashMap<Point, String> alphabet = TaskData.getAlphabet();
        HashMap<String, Object> var = TaskData.getVar();
        nB = (int)var.get("NB");
        cipherText = (List) var.get("Text");

        curve = new EllepticCurve(-1, 1, 751); // E(-1, 1) mod 751

        //дешифруем криптограмму
        System.out.println("n = " + nB);
        String res = "";

        for (int i = 0; i < cipherText.size(); i++){
            Point[] points = cipherText.get(i);

            //Pm + kPb - nB(kG) = alphabet point
            Point invKg = new Point(points[0].getX(),-points[0].getY()); // -kG
            Point invNKg = curve.mul(invKg, nB); // -nB(kG)
            Point resP = curve.sum(points[1], invNKg); // Pm + kPb - nB(kG)

            res += alphabet.get(resP);

            System.out.println("-------------------------------------------------------------------------
----");
            System.out.println("Symbol ["+points[0].toString()+""+points[1].toString()+"]; -nkG = "+
                    invNKg.toString()+"; Pm + kPb - nkg ="+resP.toString()+"; char:
"+alphabet.get(resP));
        }
        System.out.println("Result: "+res.replace("null","_"));

    }
}
```

Результаты

```
n = 41
Symbol [(283, 493)(314, 127)]; -nkG = (418, 141); Pm + kPb - nkg =(517, 640); char: null
-------------------------------------------------------------------------
Symbol [(425, 663)(561, 140)]; -nkG = (591, 196); Pm + kPb - nkg =(240, 309); char: о
-------------------------------------------------------------------------
Symbol [(568, 355)(75, 433)]; -nkG = (120, 604); Pm + kPb - nkg =(240, 442); char: п
-------------------------------------------------------------------------
Symbol [(440, 539)(602, 627)]; -nkG = (750, 750); Pm + kPb - nkg =(247, 485); char: у
-------------------------------------------------------------------------
Symbol [(188, 93)(395, 414)]; -nkG = (489, 468); Pm + kPb - nkg =(243, 664); char: с
-------------------------------------------------------------------------
Symbol [(179, 275)(25, 604)]; -nkG = (16, 416); Pm + kPb - nkg =(247, 266); char: т
-------------------------------------------------------------------------
Symbol [(72, 254)(47, 349)]; -nkG = (568, 396); Pm + kPb - nkg =(236, 39); char: и
-------------------------------------------------------------------------
Symbol [(72, 254)(417, 137)]; -nkG = (568, 396); Pm + kPb - nkg =(238, 175); char: м
-------------------------------------------------------------------------
Symbol [(188, 93)(298, 225)]; -nkG = (489, 468); Pm + kPb - nkg =(253, 540); char: ы
-------------------------------------------------------------------------
Symbol [(56, 419)(79, 111)]; -nkG = (5, 11); Pm + kPb - nkg =(236, 712); char: й
-------------------------------------------------------------------------
Result: _опустимый
```

Выводы

В ходе данной лабораторной работы был изучен алгоритм дешифрования на основании эллиптических кривых. Была написана программа, реализующая этот алгоритм и позволяющая расшифровать криптограмму, используя приведенный в задании алфавит на основе кривой $E_{751}(-1, 1)$: $y^2 = x^3 - 1x + 1 \pmod{751}$.