Федеральное государственное автономное образовательное учреждение высшего образования "Национальный исследовательский университет ИТМО"

Лабораторная работа №6

по дисципение "Информационная безопасность"

Расшифрование криптограммы на основе эллиптических кривых

Вариант 10

Выполнил: студент Саржевский И.А.

Группа: P3402

Преподаватель: к.т.н., доцент
Маркина Т.А.

г. Санкт-Петербург

2021 г.

Лабораторная работа №6

Расшифрование криптограммы на основе эллиптических кривых

Цель работы

Расшифровать текст, используя приведенный алфавит на основе кривой $E_{751}(-1, 1)$: $y^2 = x^3 - 1x + 1\ (mod\ 751)$.

Задание

Алгоритм шифрования на основе эллиптических кривых состоит из следующих шагов:

1. Случайным образом выбирается секретный ключ n_b;
2. Вычисляется публичный ключ $P_b = n_b G$;
3. Выбирается случайное число k;
4. Рассчитывается kG;
5. Рассчитывается $P_m + kP_b$, где P_m - исходное сообщение;
6. Пара $(kG; P_m + kP_b)$ составляет зашифрованное сообщение.

Чтобы расшифровать сообщение, получателю необходимо рассчитать значение $n_b kG$, и вычесть его из $P_m + kP_b$.

Разработанная программа принимает путь к yaml-файлу, в котором описаны исходные данные: секретный ключ n_b и последовательность пар точек, представляющих зашифрованное сообщение.

Исходные данные

```yaml
n: 18
dots:
  - "(179, 275), (269, 564)"
  - "(179, 275), (73, 72)"
  - "(440, 539), (189, 454)"
  - "(618, 206), (628, 458)"
  - "(568, 355), (660, 275)"
  - "(72, 254), (709, 595)"
  - "(745, 210), (12, 314)"
  - "(188, 93), (36, 664)"
  - "(618, 206), (530, 22)"
  - "(286, 136), (532, 50)"
  - "(425, 663), (660, 275)"
  - "(725, 195), (482, 230)"
```

Листинг разработанной программы

elliptic_curve.rs

```
#[path = "./point.rs"] mod point;
pub use point::Point;

static DEBUG: bool = false;

// define the elliptic curve as E(a,b) mod p
pub struct Curve {
  pub a: i64,
  pub b: i64,
  pub p: i64
}

impl Curve {
  // multiplication as a series of P + P ... + P
  pub fn mul(&self, p: &Point, m: i32) -> Point {
    if DEBUG {println!("------- {} * {}", p, m);}
    let mut res = Point { x: p.x, y: p.y };
    for i in 1..m {
      res = self.sum(&res, p);
      if DEBUG {println!("R_{} = {}", i, res);}
    }
    return res;
  }

  // calculate the sum of two point on eliptic curve
  pub fn sum(&self, p1: &Point, p2: &Point) -> Point {
    let lambda = self.get_lambda(p1, p2);
    let mut tmp = lambda * lambda - p1.x - p2.x; // x3 = lambda^2 - x1 - x2
    let r_x = if tmp >= 0 {tmp % self.p} else {(tmp % self.p) + self.p}; // imitate the mod() behaviour
    tmp = lambda * (p1.x - r_x) - p1.y; // y3 = lambda * (x1 - x3) - y1
    let r_y = if tmp >= 0 {tmp % self.p} else {(tmp % self.p) + self.p}; // imitate the mod() behaviour
    return Point { x: r_x, y: r_y };
  }

  // calculate the lambda
  fn get_lambda(&self, p1: &Point, p2: &Point) -> i64 {
    // numenator = y2 - y1 if p1 != p2 and 3x1^2 + a otherwise
    let numerator = if !p1.equal(p2) {p2.y - p1.y}
      else {3*p1.x*p1.x + self.a};
    // denominator = x2 - x1 if p1 != p2 and 2y1 otherwise
    let mut denominator = if !p1.equal(p2) {p2.x - p1.x}
      else {2 * p1.y};
    // the implementation of the inv_mod from mod_ops seems to be buggy
    // when it comes to negative values, so I use my own implementation
    // we compute the corresponding numenator for denominator by solving
    // the `i * x mod p` equation with respect to negative values
    for i in 0..self.p {
      if ((denominator * i) % self.p + if denominator < 0 {self.p} else {0}) == 1 {
          denominator = i;
          break;
      }
    }

    let res = denominator * numerator;
    if DEBUG {
      if !p1.equal(p2) {
        println!("lambda = (y2 - y1) * invmod (x2 - x1) = ({} - {}) * {} = {}",
          p2.y, p1.y, denominator, res);
      }
      else {
        println!("lambda = (3x1^2 + a) * invmod (2*y1) = {} * {} = {}",
          (3*p1.x*p1.x + self.a), denominator, res);
      }
    }
    return res;
  }
}
```

main.rs

```
mod elliptic_curve;
mod alphabet;

extern crate regex;

use regex::Regex;
use elliptic_curve::Curve;
use elliptic_curve::Point;
use getopts::Options;
use yaml_rust::YamlLoader;
use std::fs;
use std::env;

fn find_in_alphabet(p: Point) -> Option<char> {
  for point in alphabet::ALPHABET.entries() {
    let c_p = Point { x: point.1.x, y: point.1.y };
    if c_p.equal(&p) {
      return Some(*point.0);
    }
  }
  return None;
}

fn main() {
  // creating a E(-1, 1) mod 751 curve
  let curve = Curve { a: -1, b: 1, p: 751 };

  // --- Adding cmd line arguments: ------------------
  //      -f: Path to input data yaml-file
  let args: Vec<String> = env::args().collect();

  let mut opts = Options::new();
  opts.optopt("f", "file", "input file path", "input.yaml");

  let matches = match opts.parse(&args[1..]) {
    Ok(m) => { m }
    Err(f) => { panic!(f.to_string()) }
  };

  if !matches.opt_present("f") {
    println!("You must specify the input file path!");
    return;
  }
  // -------------------------------------------------
  // --- Parsing input yaml-file ---------------------
  let file_contents = fs::read_to_string(matches.opt_str("f")
                                         .unwrap()).unwrap();
  let docs = YamlLoader::load_from_str(&file_contents).unwrap();
  let doc = &docs[0];
  let n = doc["n"].as_i64().unwrap();
  println!("n = {}", n);
  // -------------------------------------------------
  // --- Parsing the textual representation of points
  let cm_regex = Regex::new(r"\((\d+), (\d+)\), \((\d+), (\d+)\)").unwrap();
  let mut input: Vec<[Point; 2]> = Vec::new();
  for point_text in doc["dots"].as_vec().unwrap() {
    let p_text = point_text.as_str().unwrap();
    let cap = cm_regex.captures(p_text).unwrap();
    let kg = Point { x: cap[1].parse::<i64>().unwrap(), y: cap[2].parse::<i64>().unwrap() };
    let pmkpb = Point { x: cap[3].parse::<i64>().unwrap(), y: cap[4].parse::<i64>().unwrap() };
    input.push([kg, pmkpb]);
  }
  // -------------------------------------------------
  // --- Decoding message ----------------------------
  let mut res = String::new();
  println!("-----------------------------------------------------------------------------------------------");
  for sym in input {
    let minus_kg = Point { x: sym[0].x, y: -sym[0].y }; // -kG = (kG.x, -kG.y)
    let n_m_kg = curve.mul(&minus_kg, n as i32); // -nkG = n * -kG
    let res_point = curve.sum(&sym[1], &n_m_kg); // original point = Pm + kPb - nkG
    let r_char = find_in_alphabet(res_point).unwrap(); // find the character by a point
```

```
72      println!("Decoding [{} {}]; -nkG = {}; Pm+kPb-nkG = {}; char = {}",
73                  sym[0], sym[1], n_m_kg, res_point, r_char);
74      println!("-----------------------------------------------------------------------------------------------");
75      res.push(r_char);
76    }
77    println!("Decoded message: {}", res);
78  }
```

Результаты работы программы

```
keker (~/code/itmo-4th-year/infosec/part2/lab6/target/debug) master ▲ 1
↳ ./lab6 -f ../../src/var10.yaml
n = 18
-----------------------------------------------------------------------------------------------
Decoding [(179, 275) (269, 564)]; -nkG = (72, 497); Pm+kPb-nkG = (237, 297); char = к
-----------------------------------------------------------------------------------------------
Decoding [(179, 275) (73, 72)]; -nkG = (72, 497); Pm+kPb-nkG = (240, 309); char = о
-----------------------------------------------------------------------------------------------
Decoding [(440, 539) (189, 454)]; -nkG = (181, 618); Pm+kPb-nkG = (238, 175); char = м
-----------------------------------------------------------------------------------------------
Decoding [(618, 206) (628, 458)]; -nkG = (120, 604); Pm+kPb-nkG = (238, 175); char = м
-----------------------------------------------------------------------------------------------
Decoding [(568, 355) (660, 275)]; -nkG = (702, 225); Pm+kPb-nkG = (247, 485); char = у
-----------------------------------------------------------------------------------------------
Decoding [(72, 254) (709, 595)]; -nkG = (745, 541); Pm+kPb-nkG = (238, 576); char = н
-----------------------------------------------------------------------------------------------
Decoding [(745, 210) (12, 314)]; -nkG = (56, 419); Pm+kPb-nkG = (236, 39); char = и
-----------------------------------------------------------------------------------------------
Decoding [(188, 93) (36, 664)]; -nkG = (499, 595); Pm+kPb-nkG = (237, 297); char = к
-----------------------------------------------------------------------------------------------
Decoding [(618, 206) (530, 22)]; -nkG = (120, 604); Pm+kPb-nkG = (228, 271); char = а
-----------------------------------------------------------------------------------------------
Decoding [(286, 136) (532, 50)]; -nkG = (665, 598); Pm+kPb-nkG = (247, 266); char = т
-----------------------------------------------------------------------------------------------
Decoding [(425, 663) (660, 275)]; -nkG = (0, 1); Pm+kPb-nkG = (240, 309); char = о
-----------------------------------------------------------------------------------------------
Decoding [(725, 195) (482, 230)]; -nkG = (440, 212); Pm+kPb-nkG = (243, 87); char = р
-----------------------------------------------------------------------------------------------
Decoded message: коммуникатор
```

Рис. 1: Результат работы программы

Вывод

В результате выполнения данной лабораторной работы был изучен алгоритм дешифрации текста на основании эллиптических кривых и реализована программа, позволяющая расшифровать криптограмму, используя приведенный алфавит на основе кривой $E_{751}(-1, 1) : y^2 = x^3 - 1x + 1 \, (mod\, 751)$.