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

Лабораторная работа №5

по дисципение "Информационная безопасность"

Шифрование открытого текста наоснове 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 №5

Шифрование открытого текста на основе эллиптических кривых

Цель работы

Зашифровать открытый текст, используя приведенный алфавит на основе кривой $E_{751}(-1, 1)$: $y^2 = x^3 - 1x + 1\ (mod\ 751)$ и генерирующей точки $G(0, 1)$.

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: текст, который необходимо закодировать, точка P_b и последовательность значений k.

Исходные данные

```yaml
T: репарация
Bx: 435
By: 663
k:
  - 12
  - 11
  - 18
  - 7
  - 16
  - 18
  - 17
  - 2
  - 3
```

Листинг разработанной программы

alphabet_generator.py

```
# This script is parsing the text file that was produced from the task pdf
# and producing the rust code that contains the definition of the alphabet
# as a Map<char, Point>. It was created because the alphabet is pretty
# big, and writing the rust module manually will be problematic, and I
# will definitely do lots of mistakes while retyping the values :)

buffer = ""
buffer += "#[path = \"./point.rs\"] mod point;\n"
buffer += "use point::Point;\n"
buffer += "use phf::phf_map;\n\n"
buffer += "pub static ALPHABET: phf::Map<char, Point> = phf_map! {\n"

with open('alphabet.txt', 'r') as d:
  lines = d.readlines()

for line in lines:
  values = line.split('\t')
  [x, y] = values[1].replace('\n', '').replace('(', '').replace(')', '').replace(' ', '').split(',')
  buffer += f" '{values[0]}' => Point {{ x: {x}, y: {y} }},\n"

buffer += "};\n"
buffer += "\n"

with open('alphabet.rs', 'w') as d:
    d.write(buffer)
```

alphabet.rs

```
#[path = "./elliptic_curve.rs"] mod elliptic_curve;
use elliptic_curve::Point;
use phf::phf_map;

pub static ALPHABET: phf::Map<char, Point> = phf_map! {
 'B' => Point { x: 67, y: 84 },
 'e' => Point { x: 99, y: 456 },
 ' ' => Point { x: 33, y: 355 },
 'C' => Point { x: 67, y: 667 },
 'f' => Point { x: 100, y: 364 },
 '!' => Point { x: 33, y: 396 },
 'D' => Point { x: 69, y: 241 },
 'g' => Point { x: 100, y: 387 },
 '"' => Point { x: 34, y: 74 },
 'E' => Point { x: 69, y: 510 },
 'h' => Point { x: 102, y: 267 },
 '#' => Point { x: 34, y: 677 },
 'F' => Point { x: 70, y: 195 },
 'i' => Point { x: 102, y: 484 },
 '$' => Point { x: 36, y: 87 },
          ... some lines omitted
};
```

point.rs

```
use std::fmt;
// simple struct that represents the point in 2D space
#[derive(PartialEq, PartialOrd, Clone, Copy)]
pub struct Point {
    pub x: i64,
    pub y: i64
}

impl Point {
  pub fn equal(&self, other: &Point) -> bool {
    return self.x == other.x && self.y == other.y;
  }
```

```
}

impl fmt::Display for Point {
  fn fmt(&self, f: &mut fmt::Formatter<'_>) -> fmt::Result {
    write!(f, "({}, {})", self.x, self.y)
  }
}

impl fmt::Debug for Point {
  fn fmt(&self, f: &mut fmt::Formatter<'_>) -> fmt::Result {
    write!(f, "({}, {})", self.x, self.y)
  }
}
```

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) {
```

```
57          println!("lambda = (y2 - y1) * invmod (x2 - x1) = ({} - {}) * {} = {}",
58            p2.y, p1.y, denominator, res);
59        }
60        else {
61          println!("lambda = (3x1^2 + a) * invmod (2*y1) = {} * {} = {}",
62            (3*p1.x*p1.x + self.a), denominator, res);
63        }
64      }
65      return res;
66    }
67  }
```

main.rs

```
1   mod elliptic_curve;
2   mod alphabet;
3
4   use elliptic_curve::Curve;
5   use elliptic_curve::Point;
6   use getopts::Options;
7   use yaml_rust::YamlLoader;
8   use std::fs;
9   use std::env;
10
11  fn main() {
12    // creating a E(-1, 1) mod 751 curve
13    let curve = Curve { a: -1, b: 1, p: 751 };
14    let g = Point { x: 0, y: 1 }; // G = (0, 1)
15
16    // --- Adding cmd line arguments: ------------------
17    //      -f: Path to input data yaml-file
18    let args: Vec<String> = env::args().collect();
19
20    let mut opts = Options::new();
21    opts.optopt("f", "file", "input file path", "input.yaml");
22
23    let matches = match opts.parse(&args[1..]) {
24      Ok(m) => { m }
25      Err(f) => { panic!(f.to_string()) }
26    };
27
28    if !matches.opt_present("f") {
29      println!("You must specify the input file path!");
30      return;
31    }
32    // -------------------------------------------------
33    // --- Parsing input yaml-file ---------------------
34    let file_contents = fs::read_to_string(matches.opt_str("f")
35                                             .unwrap()).unwrap();
36    let docs = YamlLoader::load_from_str(&file_contents).unwrap();
37    let doc = &docs[0];
38    let bx = doc["Bx"].as_i64().unwrap();
39    let by = doc["By"].as_i64().unwrap();
40    let pb = Point { x: bx, y: by };
41    let text = doc["T"].as_str().unwrap();
42    println!("Pb = {}, Message: {}", pb, text);
43    let mut k: Vec<i32> = Vec::new();
44    for c_k in doc["k"].as_vec().unwrap() {
45      k.push(c_k.as_i64().unwrap() as i32);
46    }
47    // -------------------------------------------------
48    // --- Encrypting message --------------------------
49    let mut i = 0;
50    let mut res: Vec<Point> = Vec::new();
51    println!("-------------------------------------------------");
52    for c in text.chars() {
53      let a_pm = alphabet::ALPHABET[&c]; // get the corresponding point for symbol
54      let pm = Point { x: a_pm.x, y: a_pm.y };
55      let c_k = k[i]; // get k for current symbol
56      let kg = curve.mul(&g, c_k); // kG = k * G
57      let kpb = curve.mul(&pb, c_k); // kPb = k * Pb
58      let pmkpb = curve.sum(&kpb, &pm); // Pm + kPb
```

```
59      println!("Symbol: '{}'; k = {}; Pm = {}; kPb = {}", c, c_k, pm, kpb);
60      println!("Cm = (kG, Pm+kPb) = ({}, {})", kg, pmkpb);
61      println!("-----------------------------------------------");
62      res.push(kg);
63      res.push(pmkpb);
64      i += 1;
65    }
66    println!("Encrypted message: {:?}", res);
67  }
```

Результаты работы программы

```
keker (~/code/itmo-4th-year/infosec/part2/lab5/target/debug) master
↳ ./lab5 -f ../../src/var10.yaml
Pb = (435, 663), Message: репарация
-----------------------------------------------
Symbol: 'р'; k = 12; Pm = (243, 87); kPb = (556, 64)
Cm = (kG, Pm+kPb) = ((286, 136), (277, 285))
-----------------------------------------------
Symbol: 'е'; k = 11; Pm = (234, 587); kPb = (561, 508)
Cm = (kG, Pm+kPb) = ((179, 275), (174, 494))
-----------------------------------------------
Symbol: 'п'; k = 18; Pm = (240, 442); kPb = (268, 9)
Cm = (kG, Pm+kPb) = ((618, 206), (713, 355))
-----------------------------------------------
Symbol: 'а'; k = 7; Pm = (228, 271); kPb = (8, 619)
Cm = (kG, Pm+kPb) = ((135, 82), (339, 560))
-----------------------------------------------
Symbol: 'р'; k = 16; Pm = (243, 87); kPb = (660, 117)
Cm = (kG, Pm+kPb) = ((72, 254), (49, 8))
-----------------------------------------------
Symbol: 'а'; k = 18; Pm = (228, 271); kPb = (268, 9)
Cm = (kG, Pm+kPb) = ((618, 206), (465, 643))
-----------------------------------------------
Symbol: 'ц'; k = 17; Pm = (250, 14); kPb = (146, 368)
Cm = (kG, Pm+kPb) = ((440, 539), (198, 560))
-----------------------------------------------
Symbol: 'и'; k = 2; Pm = (236, 39); kPb = (713, 590)
Cm = (kG, Pm+kPb) = ((188, 93), (138, 493))
-----------------------------------------------
Symbol: 'я'; k = 3; Pm = (257, 458); kPb = (255, 165)
Cm = (kG, Pm+kPb) = ((56, 419), (110, 425))
-----------------------------------------------
Encrypted message: [(286, 136), (277, 285), (179, 275), (174, 494), (618, 206), (713, 355), (135, 82), (339, 560), (72, 254), (49, 8)
(618, 206), (465, 643), (440, 539), (198, 560), (188, 93), (138, 493), (56, 419), (110, 425)]
```

Рис. 1: Результат работы программы

Вывод

В результате выполнения данной лабораторной работы был изучен алгоритм шифрования текста на основании эллиптических кривых и реализована программа, позволяющая зашифровать открытый текст, используя приведенный алфавит на основе кривой $E_{751}(-1, 1) : y^2 = x^3 - 1x + 1 \, (mod\, 751)$ и генерирующей точки $G(0, 1)$.