МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ

“БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ”

**ИНТЕЛЕКТУАЛЬНЫЕ ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ**

ОТЧЁТ

По лабораторной работе № \_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Выполнил:

Студент группы ИИ-22

Копанчук Евгений Романович

Проверил\_\_:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Брест – 2023

Ход работы

*Код программы*

import { Point } from "./types"

export default class EllipticCurve {

private a: number

private b: number

private p: number

A() { return this.a }

B() { return this.b }

P() { return this.p }

constructor(

a: number,

b: number,

p: number,

) {

this.a = a

this.b = b

this.p = p

}

modP(value: number, p=this.p) {

return (value >= 0) ? value % p : p - (-value % p)

}

xgcd(a: number, b: number) {

if (a === 0) {

return { gcd: b, x: 0, y: 1 }

}

const { gcd, x: x1, y: y1 } = this.xgcd(b % a, a)

const x: number = y1 - Math.floor(b / a) \* x1

const y: number = x1

return { gcd: b, x, y }

}

sum(P: Point | null, Q: Point | null): Point | null {

if (P === null && Q === null) {

return null

}

if (P === null) {

return Q

}

if (Q === null) {

return P

}

let lm

if (P.x === Q.x && P.y === Q.y) {

const demonY = this.modP(3 \* P.x \*\* 2 + this.a)

const demonX = this.modP(this.xgcd(this.modP(2 \* P.y), this.p).x)

lm = this.modP(demonY \* demonX)

} else {

const demonY = this.modP(Q.y - P.y)

const demonX = this.modP(this.xgcd(this.modP(Q.x - P.x), this.p).x)

lm = this.modP(demonY \* demonX)

}

const x = this.modP(lm \*\* 2 - P.x - Q.x)

const y = this.modP(lm \* (P.x - x) - P.y)

return { x, y } as Point

}

multiply(k: number, P: Point | null) {

let Q: Point | null = null

for (let i = 0; i < (k >>> 0).toString(2).length; i++) {

if ((k >> i) & 1) {

Q = this.sum(P, Q)

}

P = this.sum(P, P)

}

return Q

}

reverse(P: Point | null) {

return (P) ? { x: P.x, y: -P.y } : null

}

}

import { Point } from "./types"

import EllipticCurve from "./elliptic-curve"

export default class EllipticCrypt {

private E: EllipticCurve

private G: Point

private alphabet: Map<string, Point>

constructor(E: EllipticCurve, G: Point, alphabet: Map<string, Point>) {

this.E = E

this.G = G

this.alphabet = alphabet

}

private encryptLetter(k: number, Pb: Point, letter: string) {

const C1 = this.E.multiply(k, this.G)

const alphabetPoint: Point | null = this.alphabet.get(letter) || null

const C2 = this.E.sum(this.E.multiply(k, Pb), alphabetPoint)

return { C1, C2 }

}

encrypt(k: number[], Pb: Point, message: string) {

const encrypted = []

for (let i = 0; i < message.length; i++) {

if (this.alphabet.has(message.charAt(i))) {

encrypted.push(this.encryptLetter(k[i], Pb, message.charAt(i)))

}

}

return encrypted

}

private getAlphabetKeyByValue(value: Point | null) {

if (value === null) {

return null

}

for (const pair of Array.from(this.alphabet.entries())) {

const [letter, pt] = pair

if (pt.x === value.x && pt.y === value.y) {

return letter

}

}

return null

}

private decryptLetter(nb: number, encryptedPair: { C1: Point | null, C2: Point | null }) {

const { C1, C2 } = encryptedPair

const C = this.E.multiply(nb, C1)

const Cm = this.E.reverse(C)

const Ca = this.E.sum(C2, Cm)

const letter = this.getAlphabetKeyByValue(Ca)

return letter

}

decrypt(nb: number, encrypted: { C1: Point | null; C2: Point | null; }[]) {

let message = ""

for (let i = 0; i < encrypted.length; i++) {

const letter = this.decryptLetter(nb, encrypted[i])

message += (letter) ? letter : "\_"

}

return message

}

}

import { Point } from "./types"

import EllipticCurve from "./elliptic-curve"

export default class EllipticSignature {

private E: EllipticCurve

private G: Point

private n: number

constructor(E: EllipticCurve, G: Point, n: number) {

this.E = E

this.G = G

this.n = n

}

sign(k: number, e: number, d: number) {

const kG = this.E.multiply(k, this.G)

if (kG === null) {

return null

}

const r = this.E.modP(kG.x, this.n)

const z = this.E.modP(this.E.xgcd(k, this.n).x, this.n)

const s = this.E.modP(z \* (e + d \* r), this.n)

return { r, s }

}

verify(e: number, r: number, s: number, Q: Point) {

if ((r < 1 || r >= this.n) || (s < 1 || s >= this.n)) {

return null

}

const v = this.E.xgcd(s, this.n).x % this.n

const u1 = this.E.modP(e \* v, this.n)

const u2 = this.E.modP(r \* 3, this.n)

const u1G = this.E.multiply(u1, this.G)

const u2Q = this.E.multiply(u2, Q)

const pt = this.E.sum(u1G, u2Q)

if (pt === null) {

return null

}

const X = this.E.modP(pt.x, this.n)

if (r === X) {

return true

}

return false

}

}

*Вывод программы*

Task 1

[

{ C1: { x: 135, y: 82 }, C2: { x: 642, y: 698 } },

{ C1: { x: 489, y: 468 }, C2: { x: 520, y: 2 } },

{ C1: { x: 56, y: 419 }, C2: { x: 719, y: 538 } },

{ C1: { x: 346, y: 242 }, C2: { x: 25, y: 604 } },

{ C1: { x: 618, y: 206 }, C2: { x: 319, y: 233 } },

{ C1: { x: 618, y: 206 }, C2: { x: 396, y: 481 } },

{ C1: { x: 346, y: 242 }, C2: { x: 209, y: 669 } },

{ C1: { x: 179, y: 275 }, C2: { x: 250, y: 14 } },

{ C1: { x: 72, y: 254 }, C2: { x: 590, y: 375 } }

]

\_тступить

Task 2

динамометр

Task 3

{ x: 203, y: 427 }

Task 4

{ x: 589, y: 322 }

Task 5

{ r: 11, s: 8 }

Task 6

false