Министерство образования Республики Беларусь

Учреждение образования

«Брестский Государственный технический университет»

Кафедра ИИТ

**Лабораторная работа №4**

По дисциплине «Криптографические методы защиты информации»

Тема: «Криптография на эллиптических кривых»

**Выполнил:**

Студент 3 курса

Группы ИИ-21

Пучинский А.А.

**Проверил:**

Хацкевич А. С.

Брест 2023

**Цель:** научиться применять алгоритмы шифрования, основанные на эллиптических кривых.

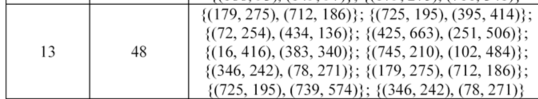
**Ход работы:**

**Вариант** 13







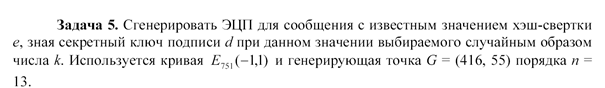


















import random as rd

class ElipticCurves:

    def \_\_init\_\_(self):

        self.p = 751

        self.a = -1

        self.b = 1

        self.G = (0, 1)

        self.Gq23 = (-1, 1)

        self.G5 = (416, 55)

        self.n = 13

        self.G6 = (562, 89)

        # n = 13

        self.G\_test = (384, 475)

        self.alphabet = {

        ' ' : (33, 355), '!' : (33, 396), '"' : (34, 74), '#' : (34, 677), '$' : (36, 87), '%' : (36, 664), '&' : (39, 171), '\'' : (39, 171), '(' : (43, 224), ')' : (43, 527),

        '\*' : (44, 366), '+' : (44, 385), ',' : (45, 31), '-' : (45, 720), '.' : (47, 349), '/' : (47, 402), '0' : (48, 49), '1' : (48, 702), '2' : (49, 183), '3' : (49, 568),

        '4' : (53,227), '5' : (53, 474), '6' : (56, 332), '7' : (56, 419), '8' : (58, 139), '9' : (58, 612), ':' : (59, 365), ';' : (59, 386), '<' : (61, 129), '=' : (61, 622),

        '>' : (62, 372), '?' : (62, 379), '@' : (66, 199), 'A' : (66, 552), 'B' : (67, 84), 'C' : (67, 667), 'D' : (69, 241), 'E' : (69, 510), 'F' : (70, 195), 'G' : (70, 556),

        'H' : (72, 254), 'I' : (72, 497), 'J' : (73, 72), 'K' : (73, 679), 'L' : (74, 170), 'M' : (74, 581), 'N' : (75, 318), 'O' : (75, 433), 'P' : (78, 271), 'Q' : (78, 480),

        'R' : (79, 111), 'S' : (79, 640), 'T' : (80, 318), 'U' : (80, 433), 'V' : (82, 270), 'W' : (82, 481), 'X' : (83, 373), 'Y' : (83, 378), 'Z' : (85, 35), '[' : (85, 716),

        '\\' : (86, 25), ']' : (86, 726), '^' : (90, 21), '\_' : (90, 730), '`' : (93, 267), 'a' : (93, 484), 'b' : (98, 338), 'c' : (98, 413), 'd' : (99, 295), 'e' : (99, 456),

        'f' : (100, 364), 'g' : (100, 387), 'h' : (102, 267), 'i' : (102, 484), 'j' : (105, 369), 'k' : (105, 382), 'l' : (106, 24), 'm' : (106, 727), 'n' : (108, 247),

        'o' : (108, 504), 'p' : (109, 200), 'q' : (109, 551), 'r' : (110, 129), 's' : (110, 622), 't' : (114, 144), 'u' : (114, 607), 'v' : (115, 242), 'w' : (115, 509),

        'x' : (116, 92), 'y' : (116, 659), 'z' : (120, 147), '{' : (120, 604), '|' : (125, 292), '}' : (125, 459), '~' : (126, 33), 'А' : (189, 297), 'Б' : (189, 458),

        'В' : (192, 32), 'Г' : (192, 719), 'Д' : (194, 205), 'Е' : (194, 546), 'Ж' : (197, 145), 'З' : (197, 606), 'И' : (198, 224), 'Й' : (198, 527), 'К' : (200, 30),

        'Л' : (200, 721), 'М' : (203, 324), 'Н' : (203, 427), 'О' : (205, 372), 'П' : (205, 379), 'Р' : (206, 106), 'С' : (206, 645), 'Т' : (209, 82), 'У' : (209, 669),

        'Ф' : (210, 31), 'Х' : (210, 720), 'Ц' : (215, 247), 'Ч' : (215, 504), 'Ш' : (218, 150), 'Щ' : (218, 601), 'Ъ' : (221, 138), 'Ы' : (221, 613), 'Ь' : (226, 9),

        'Э' : (226, 742), 'Ю' : (227, 299), 'Я' : (227, 452), 'а' : (228, 271), 'б' : (228, 480), 'в' : (229, 151), 'г' : (229, 600), 'д' : (234, 164), 'е' : (234, 587),

        'ж' : (235, 19), 'з' : (235, 732), 'и' : (236, 39), 'й' : (236, 712), 'к' : (237, 297), 'л' : (237, 454), 'м' : (238, 175), 'н' : (238, 576), 'о' : (240, 309),

        'п' : (240, 442), 'р' : (243, 87), 'с' : (243, 664), 'т' : (247, 266), 'у' : (247, 485), 'ф' : (249, 183), 'х' : (249, 568), 'ц' : (250, 14), 'ч' : (250, 737),

        'ш' : (251, 245), 'щ' : (251, 506), 'ъ' : (253, 211), 'ы' : (253, 540), 'ь' : (256, 121), 'э' : (256, 630), 'ю' : (257, 293), 'я' : (257, 458)

        }

    def extend\_eucled(self, a, b):

        if a == 0:

            return (b, 0, 1)

        gcd, x1, y1 = self.extend\_eucled(b % a, a)

        x = y1 - (b // a) \* x1

        y = x1

        return (gcd, x, y)

    def gcd(self, a, m):

        if a == 0:

            return 0

        gcd, x, y = self.extend\_eucled(a, m)

        if gcd != 1:

            print("error gcd")

            return

        return x % m

    def doubling\_additing(self, P, Q):

        if P is None:

            return Q

        if Q is None:

            return P

        x1, y1 = P

        x2, y2 = Q

        if P != Q:

            denom1 = (y2 - y1) % self.p

            denom2 = (x2 - x1) % self.p

            denom2\_ = self.gcd(denom2, self.p)

            lm = (denom1 \* denom2\_) % self.p

        elif P == Q:

            denom1 = (3 \* x1 \*\* 2 + self.a) % self.p

            denom2 = (2 \* y1) % self.p

            denom2\_ = self.gcd(denom2, self.p)

            lm = (denom1 \* denom2\_) % self.p

        x3 = (lm \*\* 2 - x1 - x2) % self.p

        y3 = (lm \* (x1 - x3) - y1) % self.p

        return (x3, y3)

    def multiply\_point(self, k, P):

        Q = None

        for i in range(k.bit\_length()):

            if (k >> i) & 1:

                Q = self.doubling\_additing(P, Q)

            P = self.doubling\_additing(P, P)

        return Q

    def encrypt\_char(self, k, Pb, char):

        C1 = self.multiply\_point(k, self.G)

        C2 = self.multiply\_point(k, Pb)

        C2\_ = self.doubling\_additing(C2, self.alphabet[char])

        C = (C1, C2\_)

        return C

    def encrypt\_text(self, k, Pb, plaintext):

        answer = []

        i = 0

        for i, char in enumerate(plaintext):

            answer.append(self.encrypt\_char(k[i], Pb, char))

            print(f"x: {answer[i][0]} k \* G, y: {answer[i][1]} Pm + k \* Pb")

        print("\n")

    def reverse\_sign(self, C):

        x, y = C

        C\_ = (x, -y)

        return C\_

    def find\_key\_by\_value(self, value):

        for key, val in self.alphabet.items():

            if val == value:

                return key

        return None

    def decrypt\_char(self, C, nb):

        C\_ = self.multiply\_point(nb, C[0])

        C\_m = self.reverse\_sign(C\_)

        C\_a = self.doubling\_additing(C[1], C\_m)

        char = self.find\_key\_by\_value(C\_a)

        if char is None:

            print("err char ind")

            return

        return char

    def decrypt\_text(self, nb, ciphertext):

        decrypted\_text = ""

        for C in ciphertext:

            char = self.decrypt\_char(C, nb)

            if char == None:

                print("error find char")

                return None

            decrypted\_text += char

        print(decrypted\_text)

        return decrypted\_text

    def signature\_generate(self, k, e, d):

        kG = self.multiply\_point(k, self.G5)

        r = kG[0] % self.n

        z = self.gcd(k, self.n)

        s = z \* (e + d \* r) % self.n

        print(f"r: {r}, s: {s}")

        return (r, s)

    def signature\_verification(self, e, rs, Q):

        if (rs[0] >= 1 and rs[0] <= self.n - 1) and (rs[1] >= 1 and rs[1] <= self.n - 1):

            v = self.gcd(rs[1], self.n)

            u1 = e \* v % self.n

            u2 = rs[0] \* 3 % self.n

            u1G = self.multiply\_point(u1, self.G6)

            # u1G = self.multiply\_point(u1, self.G\_test)

            u2Q = self.multiply\_point(u2, Q)

            X = self.doubling\_additing(u1G, u2Q)

            X\_ = X[0] % self.n

            if rs[0] == X\_:

                print(f"r: {rs[0]}, x: {X\_}")

                return True

            else:

                print(f"r: {rs[0]}, x: {X\_}")

                return False

        else:

            print("error segn ver")

            return

obj = ElipticCurves()

def task1():

    plaintext = "симмтерия"

    Pb = (179, 275)

    k = (11, 17, 18, 19, 16, 6, 12, 8, 2)

    print(obj.encrypt\_text(k, Pb, plaintext))

def task2():

    nb = 34

    cipher = [

        ((618, 206), (426, 662)),

        ((72, 254), (67, 667)),

        ((286, 136), (739, 574)),

        ((16, 416), (143, 602)),

        ((618, 206), (313, 203)),

        ((618, 206), (114, 607)),

        ((618, 206), (438, 711)),

        ((188, 93), (573, 168)),

    ]

    obj.decrypt\_text(nb, cipher)

def task3():

    P = (56, 332)

    Q = (69, 241)

    R = (83, 373)

    kP = 2

    kQ = 3

    p2 = obj.multiply\_point(kP, P)

    q3 = obj.multiply\_point(kQ, Q)

    r\_ = obj.reverse\_sign(R)

    p2q3 = obj.doubling\_additing(p2, q3)

    C = obj.doubling\_additing(p2q3, r\_)

    print(f"x: {C[0]}, y: {C[1]}")

def task4():

    P = (43, 527)

    k = 107

    C = obj.multiply\_point(k, P)

    print(f"x: {C[0]}, y: {C[1]}")

def task5():

    k = 5

    e = 9

    d = 3

    obj.signature\_generate(k, e, d)

def task6():

    e = 4

    rs = (11, 4)

    Q = (596, 318)

    obj.signature\_verification(e, rs, Q)

task1()

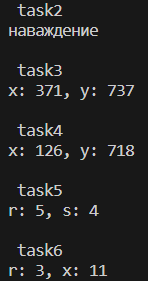
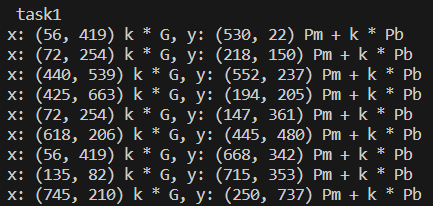
task2()

task3()

task4()

task5()

task6()



**Вывод:** в ходе лабораторной работы я научился применять алгоритмы шифрования, основанные на эллиптических кривых.