Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет информационных технологий

Кафедра информационных систем и технологий

**Лабораторная работа № 13**

«ИССЛЕДОВАНИЕ КРИПТОГРАФИЧЕСКИХ АЛГОРИТМОВ НА ОСНОВЕ ЭЛЛИПТИЧЕСКИХ КРИВЫХ»

Выполнил:

Студент Лысков К.Е.

ФИТ 3 курс 4 группа

Преподаватель: Сазонова Д.В.

Минск 2023

1. **Описание приложения**

Приложение написано на языке программирования Python и позволяет провести зашифрование и расшифрование собственной фамилии на основе ЭК.

1. **Методика выполнения расчетов**

В данной лабораторной работе была поставлена цель создания приложения, которое позволяет провести зашифрование и расшифрование собственной фамилии на основе ЭК. На листинге 2.1 представлены функции, реализующие данную функциональность.

import random

from support\_function import InverseNumber

alphabet = {'А': (189, 297), 'Б': (189, 454), '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, 601), 'Щ': (218, 601), 'Ъ': (221, 138), 'Ы': (221, 613),

            'Ь': (226, 9), 'Э': (226, 742), 'Ю': (227, 299), 'Я': (227, 452),

            }

inverse\_alphabet = {alphabet[key]: key for key in alphabet.keys()}

class EllipticCurve:

    points = []

    def \_\_init\_\_(self, a, b, p):

        self.a = a  *# Коэффициент a уравнения кривой*

        self.b = b  *# Коэффициент b уравнения кривой*

        self.p = p  *# Простое число, определяющее модуль*

    def get\_points\_by\_range(self, a, b):

        if b >= self.p:

            b = self.p

        list\_remains = []

        for i in range(a, b):

            list\_remains.append(pow(i, 2, self.p))

        list\_square\_roots = []

        for i in range(a, b):

            list\_square\_roots.append((i \*\* 3 + self.a \* i + self.b) % self.p)

        list\_points = []

        for x, i in enumerate(list\_square\_roots):

            if i in list\_remains:

                if i != 0:

                    list\_points += list((x + a, a + index)

                                        for index, j in enumerate(list\_remains) if j == i)

        return list\_points

    def add\_points\_by\_range(self, a, b):

        self.points += self.get\_points\_by\_range(a, b)

    def is\_point\_on\_curve(self, point):

        return (point[1] \*\* 2) % self.p == (point[0] \*\* 3 + self.a \* point[0] + self.b) % self.p

    def point\_addition(self, point\_1, point\_2):

        if point\_1[0] == point\_2[0] and point\_1[1] == point\_2[1]:

            s = (3 \* (point\_1[0] \*\* 2) + self.a) \* \

                pow(2 \* point\_1[1], -1, self.p) % self.p

        else:

            s = (point\_2[1] - point\_1[1]) \* \

                pow(point\_2[0] - point\_1[0], -1, self.p) % self.p

        x3 = (s \*\* 2 - point\_1[0] - point\_2[0]) % self.p

        y3 = (s \* (point\_1[0] - x3) - point\_1[1]) % self.p

        return x3, y3

    def point\_multiplication(self, point, n):

        result\_x = None

        result\_y = None

        current\_x = point[0]

        current\_y = point[1]

        while n > 0:

            if n & 1 == 1:

                if result\_x is None:

                    result\_x = current\_x

                    result\_y = current\_y

                else:

                    result\_x, result\_y = self.point\_addition(

                        (result\_x, result\_y), (current\_x, current\_y))

            current\_x, current\_y = self.point\_addition(

                (current\_x, current\_y), (current\_x, current\_y))

            n >>= 1

        return result\_x, result\_y

    def point\_difference(self, point\_1, point\_2):

        return self.point\_addition(point\_1, (point\_2[0], -point\_2[1]))

def task\_1():

    curve = EllipticCurve(-1, 1, 751)

    a = 0

    b = 750

    curve.add\_points\_by\_range(a, b)

    a = 106

    b = 140

    k = 9

    r = 7

    P = (56, 332)

    Q = (69, 241)

    R = (83, 373)

    print(f'----- Задание 1. Нахождение точек на диапазоне и операции с точками -----\n'

          f'--- 1.1 --- Точки на ЭК E751(-1, 1) в диапазоне {a, b}: {curve.get\_points\_by\_range(a, b)}\n'

          f'Количество точек в диапазоне {a, b}: {len(curve.get\_points\_by\_range(a, b))}\n'

          f'--- 1.2 ---\n'

          f'а) kР = {curve.point\_multiplication(P, k)}\n'

          f'б) P + Q = {curve.point\_addition(P, Q)}\n'

          f'в) kP + rQ - R = {curve.point\_difference(curve.point\_addition(curve.point\_multiplication(P, k), curve.point\_multiplication(Q, r)), R)}\n'

          f'г) Р – Q + R = {curve.point\_addition(curve.point\_difference(P, Q), R)}')

def EncryptByEC(message: str):

    curve = EllipticCurve(-1, 1, 751)

    message\_EC = [alphabet[i] for i in message.upper()]

    G = (0, 1)

    d = 12

    Q = curve.point\_multiplication(G, d)

    k = int(random.random() \* curve.p)

*# Зашифровывание*

    encrypt\_message\_EC = []

    for i in message\_EC:

        c\_1 = curve.point\_multiplication(G, k)

        c\_2 = curve.point\_addition(i, curve.point\_multiplication(Q, k))

        encrypt\_message\_EC.append((c\_1, c\_2))

*# Расшифровывание*

    decrypt\_message\_EC = []

    decrypt\_message = ''

    for i in encrypt\_message\_EC:

        m = curve.point\_difference(i[1], curve.point\_multiplication(i[0], d))

        decrypt\_message\_EC.append(m)

        decrypt\_message += inverse\_alphabet[m]

    print(f'\n----- Задание 2.  Зашифровывание/расшифровывание информации с помощью ЭК -----\n'

          f'G = {G}\n'

          f'd = {d}\n'

          f'Q = {Q}\n'

          f'k = {k}\n'

          f'Сообщение: {message}\n'

          f'Сообщение ЭК: {message\_EC}\n'

          f'ЭК имеет вид: \n'

          f'Зашифрованное сообщение ЭК: {encrypt\_message\_EC}\n'

          f'Расшифрованное сообщение ЭК: {decrypt\_message\_EC}\n'

          f'Расшифрованное сообщение: {decrypt\_message}\n')

    pass

def EDSBasedOnEC(message: str):

    curve = EllipticCurve(-1, 1, 751)

    G = (416, 55)

    q = 13

    s = 0

    r = 0

    u\_1 = 0

    u\_2 = 0

    while s == 0 or r == 0 or u\_1 == 0 or u\_2 == 0:

        d = 12

        k = 8

        Q = curve.point\_multiplication(G, d)

*# Отправитель*

        r = curve.point\_multiplication(G, k)[0] % curve.p

        t = InverseNumber(k, q)[1] % q

        hash\_message\_sender = alphabet[message[0].upper()][0] % 13

        s = (t \* (hash\_message\_sender + d \* r)) % q

*# Получатель*

        result\_condition\_fulfillment = r > 1 and s < q

        hash\_recipient = alphabet[message[0].upper()][0] % 13

        w = InverseNumber(s, q)[1] % q

        u\_1 = pow(w \* hash\_recipient, 1, q)

        u\_2 = pow(w \* r, 1, q)

        v = curve.point\_addition(curve.point\_multiplication(

            G, u\_1), curve.point\_multiplication(Q, u\_2))[0]

        result\_verification = r == v

    print(f'----- Задание 3. Генерация ЭЦП на основе ECDSA -----\n'

          f'G = {G}\n'

          f'q = {q}\n'

          f'd = {d}\n'

          f'k = {k}\n'

          f'Q = {Q}\n'

          f'r = {r}\n'

          f't = {t}\n'

          f'H(message) = {hash\_message\_sender}\n'

          f's = {s}\n'

          f'Отправлено {message} и ЭЦП({r, s})\n')

    print(f'Условие 1 < r, s < q {"выполняется!" if result\_condition\_fulfillment else "не выполняется!"}\n'

          f'Хеш получателя = {hash\_recipient}\n'

          f'w = {w}\n'

          f'u\_1 = {u\_1}\n'

          f'u\_2 = {u\_2}\n'

          f'v = {v}\n'

          f'Сравниваем v и r, т.е. {v} и {r}\n'

          f'Подпись верифицирована {"успешно!" if result\_verification else "не успешно!"}')

    pass

def InverseNumber(a: int, N: int) -> int:

    if N == 0:

        return a, 1, 0

    else:

        d, x, y = InverseNumber(N, a % N)

        return d, y, x - y \* (a // N)

Листинг 2.1 –код программы, реализующие заданную ранее функциональность

**3. Результаты работы приложения**

Для выполнения расчетов достаточно необходимо запустить приложение. Рисунок 3.1 и 3.2 показывают требуемые в данной лабораторной работе результаты.

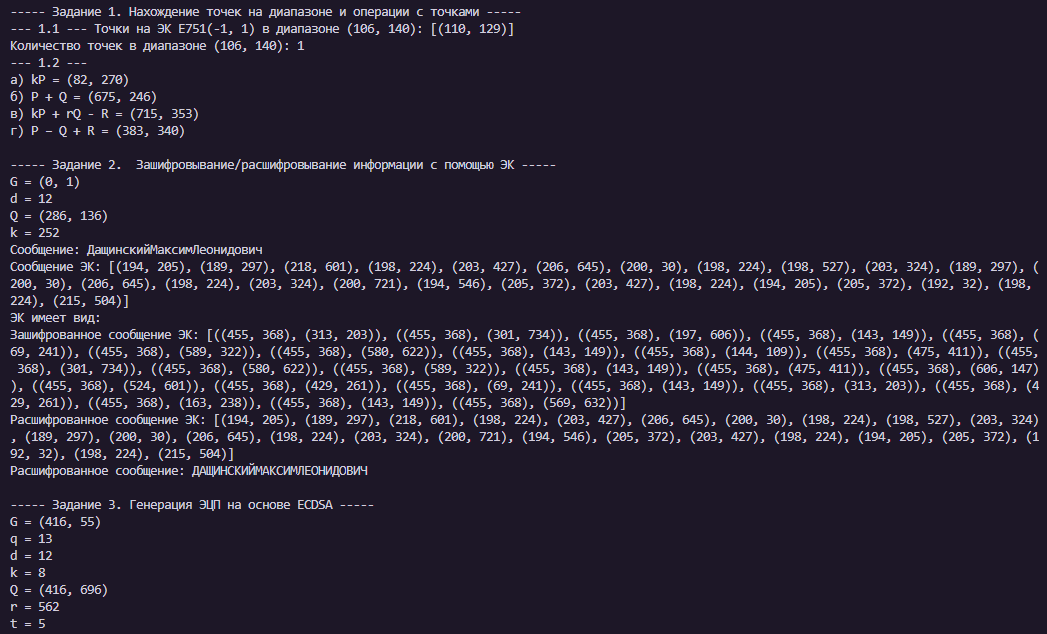


Рисунок 3.1 – Результат работы

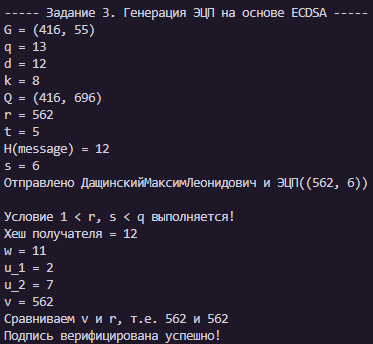


Рисунок 3.2 – Результат работы

**Вывод**

В ходе лабораторной работы было разработано приложение для зашифрования и расшифрования собственной фамилии на основе ЭК.