**ЛАБОРАТОРНАЯ РАБОТА №5**

“Шифрование открытого текста на основе эллиптических кривых”

по дисциплине

‘Информационная безопасность’

Вариант 15

***Выполнил:***

Соболев Иван Александрович

**Группа:** P34312

***Преподаватель:***

Маркина Татьяна Анатольевна

Санкт-Петербург, 2024

# Цель работы

# Зашифровать открытый текст, используя приведенный алфавит на основе кривой 𝐸751(−1, 1) ∶ 𝑦2 = 𝑥3 − 1𝑥 + 1 (𝑚𝑜𝑑 751) и генерирующей точки 𝐺(0, 1).

# Программные и аппаратные средства

Для выполнения лабораторной работы был использован компьютер со следующими характеристиками:

* Процессор: Apple M2
* Видеокарта: Apple M2
* Объем оперативной памяти: 8GB
* Использована операционная система: macOS 14.4.1
* Версия Python: 3.13

# Задание

|  |  |  |  |
| --- | --- | --- | --- |
| **№**  **варианта** | **Открытый**  **текст** | **Открытый**  **ключ *B*** | **Значения случайных чисел *k* для**  **букв открытого текста** |
| 15 | отставной | (286, 136) | 5, 3, 3, 2, 4, 19, 2, 4, 10 |

# Листинг разработанной программы

main.py

import sys  
  
import alphabet  
from curve import Curve  
from io\_utils import print\_red, read\_config, print\_separator  
from point import Point  
  
  
def main():  
 curve = Curve(-1, 1, 751)  
 g = Point(0, 1)  
 if len(sys.argv) < 2:  
 print\_red("Укажите название файла с параметрами!")  
 return  
  
 input\_file = None  
 for i in range(1, len(sys.argv)):  
 if sys.argv[i] == '-f' and i + 1 < len(sys.argv):  
 input\_file = sys.argv[i + 1]  
  
 doc = read\_config(input\_file)  
 bx = doc['Bx']  
 by = doc['By']  
 pb = Point(bx, by)  
 text = doc['T']  
 print(f"Pb = {pb}, Сообщение: {text}")  
  
 k = [int(c\_k) for c\_k in doc['k']]  
  
 res = []  
 print\_separator()  
 for i, c in enumerate(text):  
 a\_pm = alphabet.ALPHABET[c]  
 pm = Point(a\_pm.x, a\_pm.y)  
 c\_k = k[i]  
 kg = curve.elliptic\_mul(g, c\_k)  
 kpb = curve.elliptic\_mul(pb, c\_k)  
 pmkpb = curve.elliptic\_add(kpb, pm)  
 print(f"Исходный символ: '{c}'; k = {c\_k}; Pm = {pm}; kPb = {kpb}")  
 print(f"Cm = (kG, Pm+kPb) = ({kg}, {pmkpb})")  
 print\_separator()  
 res.append(kg)  
 res.append(pmkpb)  
  
 print(f"Зашифрованное сообщение: ")  
 print(res, sep='\n')  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

by = doc['By']  
 pb = Point(bx, by)  
 text = doc['T']  
 print(f"Pb = {pb}, Сообщение: {text}")  
  
 k = [int(c\_k) for c\_k in doc['k']]  
  
 res = []  
 print\_separator()  
 for i, c in enumerate(text):  
 a\_pm = alphabet.ALPHABET[c]  
 pm = Point(a\_pm.x, a\_pm.y)  
 c\_k = k[i]  
 kg = curve.elliptic\_mul(g, c\_k)  
 kpb = curve.elliptic\_mul(pb, c\_k)  
 pmkpb = curve.elliptic\_add(kpb, pm)  
 print(f"Исходный символ: '{c}'; k = {c\_k}; Pm = {pm}; kPb = {kpb}")  
 print(f"Cm = (kG, Pm+kPb) = ({kg}, {pmkpb})")  
 print\_separator()  
 res.append(kg)  
 res.append(pmkpb)  
  
 print(f"Зашифрованное сообщение: ")  
 print(res, sep='\n')  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

Io\_utils.py

import yaml  
  
  
def read\_config(input\_file):  
 *"""Метод чтения параметров из конфигурационного файла."""* with open(input\_file, 'r') as file:  
 file\_contents = file.read()  
  
 return yaml.safe\_load(file\_contents)  
  
  
def print\_green(message: str) -> None:  
 *"""Метод для вывода ключа"""* print(f"\033[92m{message}\033[0m")  
  
  
def print\_red(message: str) -> None:  
 *"""Метод для вывода расшифрованного текста"""* print(f"\033[91m{message}\033[0m")  
  
  
def print\_separator():  
 *"""Метод для вывода разграничителя"""* print\_green("-----------------------------------------------------------------")

def print\_separator():  
 *"""Метод для вывода разграничителя"""* print\_green("-----------------------------------------------------------------")

point.py

class Point:  
 def \_\_init\_\_(self, x: int, y: int):  
 self.x = x  
 self.y = y  
  
 def \_\_str\_\_(self) -> str:  
 return f"({self.x}, {self.y})"  
  
 def \_\_repr\_\_(self) -> str:  
 return f"({self.x}, {self.y})"

curve.py

from point import Point  
  
  
class Curve:  
 def \_\_init\_\_(self, a: int, b: int, p: int):  
 self.a = a  
 self.b = b  
 self.p = p  
  
 def elliptic\_mul(self, p: Point, k):  
 *"""Метод эллиптического умножения"""* R = p  
 for i in range(1, k):  
 R = self.elliptic\_add(R, p)  
  
 return R  
  
 def elliptic\_add(self, P1: Point, P2: Point):  
 *"""Метод эллиптического сложения"""* x1, y1 = P1.x, P1.y  
 x2, y2 = P2.x, P2.y  
  
 if P1 != P2:  
 slope = (y2 - y1) \* pow(x2 - x1, -1, self.p)  
 else:  
 slope = (3 \* x1 \*\* 2 + self.a) \* pow(2 \* y1, -1, self.p)  
  
 x3 = pow(slope \* slope - x1 - x2, 1, self.p)  
 y3 = pow(slope \* (x1 - x3) - y1, 1, self.p)  
  
 return Point(x3, y3)

alphabet.py

from point import Point  
  
ALPHABET = {  
 'B': Point(67, 84),  
 'e': Point(99, 456),  
 'Й': Point(198, 527),  
 ' ': Point(33, 355),  
 'C': Point(67, 667),  
 'f': Point(100, 364),  
 'К': Point(200, 30),  
 …

config.yaml

T: отставной  
Bx: 286  
By: 136  
k:  
 - 5  
 - 3  
 - 3  
 - 2  
 - 4  
 - 19  
 - 2  
 - 4  
 - 10

# Результаты работы программы

A screenshot of a computer

Description automatically generated