Министерство образования Республики Беларусь

Учреждение образования

«Брестский государственный технический университет»

Кафедра ИИТ

Лабораторная работа №4

По дисциплине: «Криптографические методы защиты информации»

**Тема: «Эллиптические кривые»**

Выполнила:

Студент 3 курса

Группы ИИ-23

Макаревич Н.Р.

Проверила:

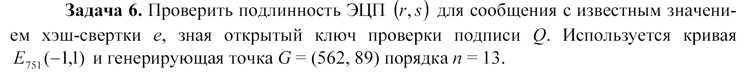
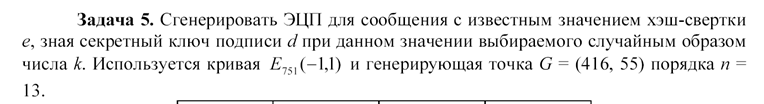
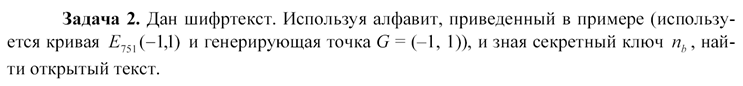
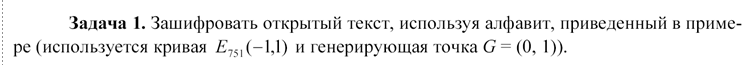
Хацкевич А. С.

Брест 2024

**Цель:** создать программу, которая реализует работу с эллиптическими кривыми

**Ход работы:**

**Вариант 3**



**Задание:**

class Curve:

def \_\_init\_\_(self, a, b, p, gen\_point):

self.a = a

self.b = b

self.p = p

self.G = gen\_point

self.alphabet = {'р': (243, 87),

'е': (234, 587),

'г': (229, 600),

'с': (243, 664),

'о': (240, 309)}

def sum\_points(self, point\_A, point\_B):

if point\_A is None:

return point\_B

if point\_B is None:

return point\_A

lmb = self.lamb(point\_A, point\_B)

x1, y1 = point\_A

x2, y2 = point\_B

x = (lmb \*\* 2 - x1 - x2) % self.p

y = (lmb \* (x1 - x) - y1) % self.p

return (x, y)

def lamb(self, point\_A, point\_B):

x1, y1 = point\_A

x2, y2 = point\_B

if point\_A == point\_B:

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

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

else:

numerator = (y2 - y1) % self.p

denominator = (x2 - x1) % self.p

den\_inv = pow(denominator, -1, self.p)

return (numerator \* den\_inv) % self.p

def scalar\_prod\_point(self, n: int, point: tuple):

temp = point

for i in range(n - 1):

point = self.sum\_points(point, temp)

return point

def subtract\_points(self, point\_A, point\_B):

x, y = point\_B

return self.sum\_points(point\_A, (x,-y))

def encrypt(self, message, open\_key, k\_arr):

cipher = []

for ind, char in enumerate(message):

Pm = self.alphabet.get(char, (None, None))

kPb = self.scalar\_prod\_point(k\_arr[ind], open\_key)

cipher.append((self.scalar\_prod\_point(k\_arr[ind], self.G), self.sum\_points(Pm, kPb)))

return cipher

def decrypt(self, cipher, key):

decrypted\_text = []

for ind, cip\_char in enumerate(cipher):

kG, PmkPb = cip\_char

kG = self.scalar\_prod\_point(key, kG)

decrypted\_text.append(self.subtract\_points(PmkPb, kG))

return decrypted\_text

def extended\_gcd(self, a, b):

if b == 0:

return a, 1, 0

g, x, y = self.extended\_gcd(b, a % b)

return g, y, x - (a // b) \* y

def mod\_inverse(self, a, p):

g, x, y = self.extended\_gcd(a, p)

if g != 1:

return None

else:

return x % p

def DSA(self, e, d, k, n):

x, y = self.scalar\_prod\_point(k, self.G)

r = x % n

s = (self.mod\_inverse(k, n) \* (e + d \* r)) % n

return (r, s)

def verify\_DSA(self, e, open\_key, n, DSA) -> bool:

r, s = DSA

if r < 1 or r > n - 1:

return False

if s < 1 or s > n - 1:

return False

v = self.mod\_inverse(s, n)

u1 = (e \* v) % n

u2 = (r \* v) % n

x, y = self.sum\_points(self.scalar\_prod\_point(u1, self.G), self.scalar\_prod\_point(u2, open\_key))

return r == x % n

**Вывод:** создал программу, которая реализует работу с эллиптическими кривыми