Министерство образования Республики Беларусь  
Учреждение образования  
«Брестский государственный технический университет»  
Кафедра ИИТ

Лабораторная работа №4  
По дисциплине: «Криптографические методы защиты информации»

Тема: «Эллиптические кривые»

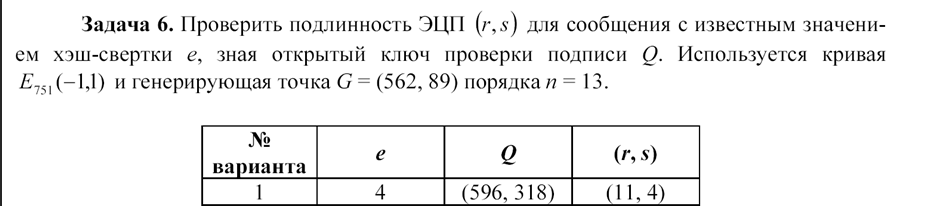
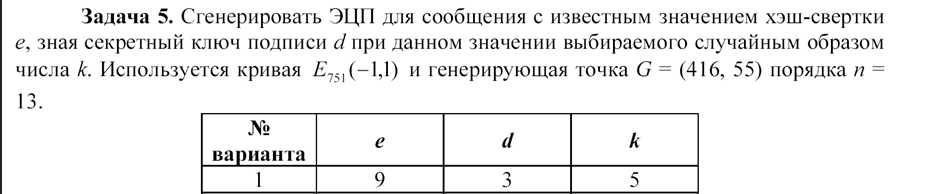
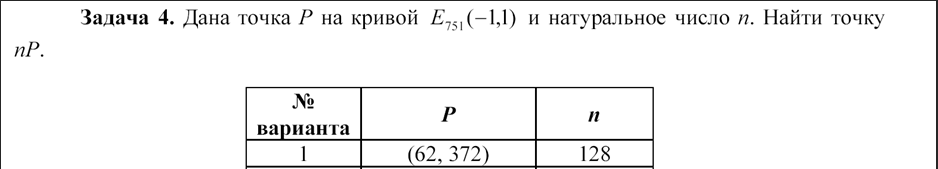
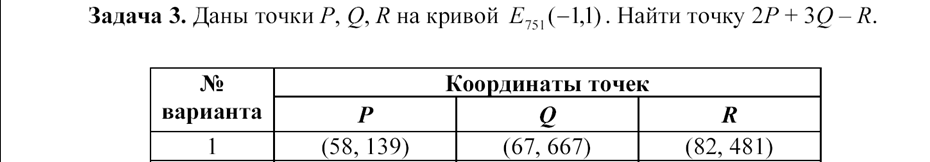
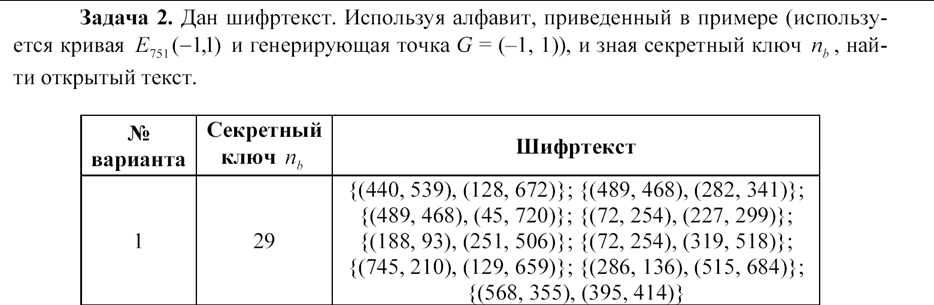
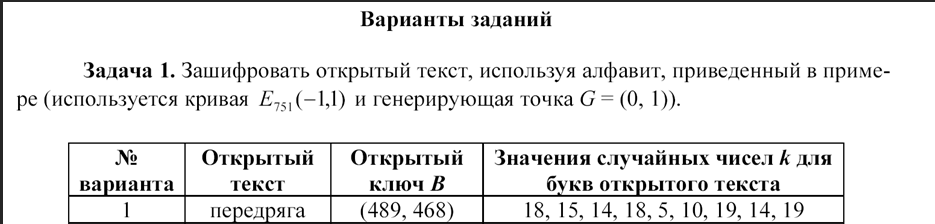
Выполнил:  
студент 3 курса  
группы ИИ-21  
Кирилович А. А.

Проверил:  
Хацкевич А. С.

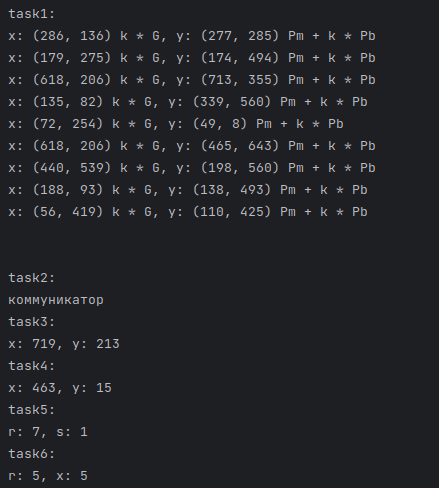
Брест 2023

Цель работы: разработать программу для упражнений на эллиптических кривых.

Вариант 10



Реализация:

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)  
 self.G\_test = (384, 475)  
 self.alphabet = {}  
 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])  
 return C1, C2\_  
 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  
 return x, -y  
 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("error 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 is 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 (1 <= rs[0] <= self.n - 1) and (1 <= 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)  
 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 sign ver")  
 return  
class Task1:  
 def \_\_init\_\_(self, plaintext: str, Pb: tuple, k\_to\_char: tuple):  
 self.plaintext = plaintext  
 self.Pb = Pb  
 self.k\_to\_char = k\_to\_char  
 def \_\_call\_\_(self, obj: ElipticCurves):  
 obj.encrypt\_text(self.k\_to\_char, self.Pb, self.plaintext)  
class Task2:  
 def \_\_init\_\_(self, nb: int, ciphertext: list[tuple[tuple]]):  
 self.nb = nb  
 self.ciphertext = ciphertext  
 def \_\_call\_\_(self, obj: ElipticCurves):  
 obj.decrypt\_text(self.nb, self.ciphertext)  
class Task3:  
 def \_\_init\_\_(self, P: tuple, Q: tuple, R: tuple, kP=2, kQ=3):  
 self.P, self.Q, self.R, self.kP, self.kQ = P, Q, R, kP, kQ  
 def \_\_call\_\_(self, obj: ElipticCurves):  
 p2 = obj.multiply\_point(self.kP, self.P)  
 q3 = obj.multiply\_point(self.kQ, self.Q)  
 r\_ = obj.reverse\_sign(self.R)  
 p2q3 = obj.doubling\_additing(p2, q3)  
 C = obj.doubling\_additing(p2q3, r\_)  
 print(f"x: {C[0]}, y: {C[1]}")  
class Task4:  
 def \_\_init\_\_(self, P: tuple, k: int):  
 self.P, self.k = P, k  
 def \_\_call\_\_(self, obj: ElipticCurves):  
 C = obj.multiply\_point(self.k, self.P)  
 print(f"x: {C[0]}, y: {C[1]}")  
class Task5:  
 def \_\_init\_\_(self, e: int, d: int, k: int):  
 self.e, self.d, self.k = e, d, k  
 def \_\_call\_\_(self, obj: ElipticCurves):  
 obj.signature\_generate(self.k, self.e, self.d)  
class Task6:  
 def \_\_init\_\_(self, e: int, Q: tuple, rs: tuple):  
 self.e, self.Q, self.rs = e, Q, rs  
  
 def \_\_call\_\_(self, obj: ElipticCurves):  
 obj.signature\_verification(self.e, self.rs, self.Q)  
def main():  
 obj = ElipticCurves()  
 print("task1:")  
 task1 = Task1(plaintext="репарация",Pb=(435, 663),k\_to\_char=(12, 11, 18, 7, 16, 18, 17, 2, 3))  
 task1(obj)  
 print("task2:")  
 task2 = Task2(nb=18,  
 ciphertext=[  
 ((179, 275), (269, 564)),  
 ((179, 275), (73, 72)),  
 ((440, 539), (189, 454)),  
 ((618, 206), (628, 458)),  
 ((568, 355), (660, 275)),  
 ((72, 254), (709, 595)),  
 ((745, 210), (12, 314)),  
 ((188, 93), (36, 664)),  
 ((618, 206), (530, 22)),  
 ((286, 136), (532, 50)),  
 ((425, 663), (660, 275)),  
 ((725, 195), (482, 230))])  
 task2(obj)  
 print("task3:")  
 task3 = Task3(P=(72, 497),Q=(53, 474),R=(90, 730))  
 task3(obj)  
 print("task4:")  
 task4 = Task4(P=(78, 480), k=147)  
 task4(obj)  
 print("task5:")  
 task5 = Task5(e=3,d=3,k=11)  
 task5(obj)  
 print("task6:")  
 task6 = Task6(e=7,Q=(384, 475),rs=(5, 5))  
 task6(obj)  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

openssl ecparam -name prime256v1 -genkey -out privatekey.pem

openssl req -new -key privatekey.pem -x509 -out certificate.pem -subj "/CN=myserver"

echo -n "Введите данные для подписи: " && read data

echo -n "$data" | openssl dgst -sha256 -sign privatekey.pem -out signature.bin

openssl x509 -in certificate.pem -pubkey -noout > pubkey.pem

echo -n "Введите данные, подписанные ранее: " && read original\_data

echo -n "$original\_data" | openssl dgst -sha256 -verify pubkey.pem -signature signature.bin

Вывод: разработал программу для упражнений на эллиптических кривых.