

НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО
ФАКУЛЬТЕТ ПРОГРАММНОЙ ИНЖЕНЕРИИ И КОМПЬЮТЕРНОЙ ТЕХНИКИ
НАПРАВЛЕНИЕ ПРОГРАММНАЯ ИНЖЕНЕРИЯ
ОБРАЗОВАТЕЛЬНАЯ ПРОГРАММА СИСТЕМНОЕ И ПРИКЛАДНОЕ
ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ
СПЕЦИАЛИЗАЦИЯ СИСТЕМНОЕ ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 2.5
курса «Информационная безопасность»
по теме: «Шифрование открытого текста на основе
эллиптических кривых»
Вариант № 25

Выполнил студент:
Тюрин Иван Николаевич
группа: Р33102

Преподаватель:
Маркина Т.А.,
Рыбаков С.Д.

Санкт-Петербург, 2025 г.

Содержание

Лабораторная работа № 2.5. Шифрование открытого текста на основе эллиптических кривых	2
1. Описание	2
2. Выполнение задания	3
3. Вывод	6

Лабораторная работа № 2.5

Шифрование открытого текста на основе эллиптических кривых

1. Описание

Цель работы: зашифровать открытый текст, используя алфавит, приведенный в [4], в подразделе «Задачи к лабораторным работам по криптографии на эллиптических кривых (используется кривая $E_{751}(-1, 1)$ – и генерирующая точка $G = (0, 1)$)».

Порядок выполнения работы:

- ознакомьтесь с теорией в учебном пособии «Криптография», а также в учебно-методическом пособии к выполнению лабораторного практикума по дисциплине «Криптография»;
- получите вариант задания у преподавателя;
- зашифруйте открытый текст;
- результаты и промежуточные вычисления оформить в виде отчета.

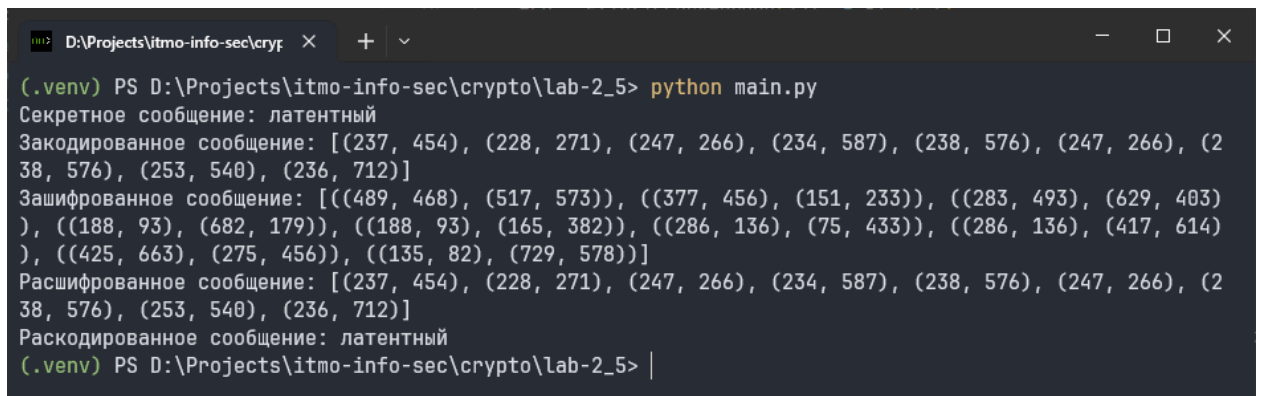
Алфавит представляет собой множество символов языка открытых текстов и соответствующих им текстов эллиптической кривой над конечным полем.

Для заданий лабораторной работы выбрана кривая $E_{751}(-1, 1)$, т.е. $y^2 = x^3 - x + 1 \pmod{751}$. Предлагается следующий (один из возможных) алфавит, приведенный в таблице. Задание варианта № 25:

№ варианта	Открытый текст	Открытый ключ В	Значения случайных чисел k для букв открытого текста
25	латентный	(725, 195)	9, 10, 13, 2, 2, 12, 12, 5, 7

2. Выполнение задания

Был реализован алгоритм выполняющий кодирование и шифрование текстового сообщения методом эллиптических кривых с заданной генерирующей точкой и набором случайных чисел, его код можно видеть на листинге 1.1. Так же был реализован алгоритм дешифрования сообщения и декодирования, который использовался для проверки корректности работы алгоритма шифрования. Кроме того, для дешифрования требуется секретный ключ, который не указан в задании варианта, поэтому в программе он получается путем перебора.



```
(.venv) PS D:\Projects\itmo-info-sec\crypto\lab-2_5> python main.py
Секретное сообщение: латентный
Закодированное сообщение: [(237, 454), (228, 271), (247, 266), (234, 587), (238, 576), (247, 266), (238, 576), (253, 540), (236, 712)]
Зашифрованное сообщение: [((489, 468), (517, 573)), ((377, 456), (151, 233)), ((283, 493), (629, 403)), ((188, 93), (682, 179)), ((188, 93), (165, 382)), ((286, 136), (75, 433)), ((286, 136), (417, 614)), ((425, 663), (275, 456)), ((135, 82), (729, 578))]
Расшифрованное сообщение: [(237, 454), (228, 271), (247, 266), (234, 587), (238, 576), (247, 266), (238, 576), (253, 540), (236, 712)]
Раскодированное сообщение: латентный
(.venv) PS D:\Projects\itmo-info-sec\crypto\lab-2_5> |
```

Рис. 1.1: Результат работы утилиты

```
1 from sympy import mod_inverse
2 from random import randint
3
4 type FP = tuple[int, int]
5 type O = tuple[None, None]
6 type Point = FP | O # finite or infinity
7
8
9 class EllipticCurveGroup:
10     def __init__(self, mod: int, a: int, b: int):
11         self.a: int = a
12         self.b: int = b
13         self.mod: int = mod
14
15     def sum(self, P: Point, Q: Point) -> Point:
16         if P == (None, None):
17             return Q
18         if Q == (None, None):
19             return P
20
21         x1, y1 = P
22         x2, y2 = Q
23
24         mod = self.mod
25         a = self.a
```

```

26
27     if x1 % mod == x2 % mod and y1 % mod == -y2 % mod:
28         return (None, None)
29
30     if P == Q:
31         s = (3 * x1**2 + a) * mod_inverse(2 * y1, mod) %
mod
32     else:
33         s = (y2 - y1) * mod_inverse(x2 - x1, mod) % mod
34
35     x3 = (s**2 - x1 - x2) % mod
36     y3 = (s * (x1 - x3) - y1) % mod
37
38     return (x3, y3)
39
40     def mul(self, k: int, P: Point) -> Point:
41         Q = (None, None)
42         N = P
43
44         while k:
45             if k & 1:
46                 Q = self.sum(Q, N)
47                 N = self.sum(N, N)
48                 k >>= 1
49
50         return Q
51
52     def encrypt(
53         self, msg: Point, G: Point, pub_key: Point, rand_k: int
54     ) -> tuple[Point, Point]:
55         C1: Point = self.mul(rand_k, G)
56         C2: Point = self.sum(msg, self.mul(rand_k, pub_key))
57         return C1, C2
58
59     def decrypt(self, C1, C2, priv_key: int) -> Point:
60         nkG: Point = self.mul(priv_key, C1)
61         neg_nkG: Point = (nkG[0], -nkG[1] % self.mod)
62         msg: Point = self.sum(C2, neg_nkG)
63         return msg
64
65
66     def read_code_table(name) -> tuple[dict, dict]:
67         encoding = {}
68         with open(name, "r", encoding="utf-8") as f:
69             for line in f.readlines():
70                 if line.strip() == "":
71                     continue
72                 i, char, *point = line.split()
73                 point = tuple(map(int, "".join(point).strip("()").
split(",")
74                 if char == "пробел":
75                     char = " "

```

```

76         encoding[char] = {"id": int(i), "code": point}
77
78     decoding = {v["code"]: k for k, v in encoding.items()}
79     return encoding, decoding
80
81
82 def write_code_table(encoding):
83     with open("code_table.txt", "w", encoding="utf-8") as f:
84         for k, v in sorted(encoding.items(), key=lambda e: e
85                               [1]["id"]):
86             if k == " ":
87                 k = "пробел"
88             f.write(f"{v['id']} {k} {v['code']}\n")
89
90 def hack_ecc(ecg: EllipticCurveGroup, G: Point, pub_key: Point)
91     -> int:
92     for k in range(ecg.mod):
93         if pub_key == ecg.mul(k, G):
94             return k
95
96 def gen_rand():
97     nums = [9, 10, 13, 2, 2, 12, 12, 5, 7]
98     for r in nums:
99         yield r
100
101     print("[random sequence end]")
102
103     while True:
104         yield randint(1, 100)
105
106
107 def main():
108     ecg = EllipticCurveGroup(751, a=-1, b=1)
109     G = (0, 1)
110     pub_key = (725, 195) # or (406,397) in example
111     text = "ЛАТЕНТНЫЙ"
112
113     secret = hack_ecc(ecg, G, pub_key) # or 45 in example
114
115     encoding, decoding = read_code_table("code_table.txt")
116     print("Секретное сообщение:", text)
117
118     encoded_text = [encoding[c]["code"] for c in text]
119     print("Закодированное сообщение:", encoded_text)
120
121     cipher = [ecg.encrypt(p, G, pub_key, k) for p, k in zip(
122         encoded_text, gen_rand())]
123     print("Зашифрованное сообщение:", cipher)
124
125     decrypted_text = [ecg.decrypt(C1, C2, secret) for C1, C2 in

```

```

125     cipher]
126     print("Расшифрованное сообщение:", decrypted_text)
127
128     decoded_text = [decoding[p] for p in decrypted_text]
129     print("Раскодированное сообщение:", "".join(decoded_text))
130
131 if __name__ == "__main__":
132     main()

```

Листинг 1.1: Код скрипта выполняющего шифрование текста методом эллиптических кривых и расшифрование тем же методом с подбором секретного ключа

3. Вывод

В результате выполнения работы изучили метод шифрования и расшифрования с помощью эллиптических кривых. Реализовали необходимые для этого алгоритмы на языке Python.

Так же было замечено, что при использовании плохого (предсказуемого) генератора случайных чисел при шифровании сообщений, они могут подвергнуться частотному анализу. Так же, как можно видеть, малые размеры конечного поля, над которым задана эллиптическая кривая позволяют найти секретный ключ по публичному достаточно быстро.