Министерство образования Республики Беларусь

Учреждение образования БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Методы защиты информации

Отчёт по лабораторной работе №4

По теме «Асимметричная криптография. Алгоритм Мак-Элиса криптография»

Выполнили: студент гр. 053501 Кривецкий Р. А.

Проверил: ассистент кафедры информатики Лещенко Е. А.

Минск 2023

**СОДЕРЖАНИЕ**

[1 Теоретические сведения 3](#_Toc148061125)

[2 Программная реализация 4](#_Toc148061126)

[Заключение 5](#_Toc148061127)

[Приложение А(обязательное) Листинг программного кода 6](#_Toc148061128)

# 1 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

McEliece — криптосистема с открытыми ключами на основе теории алгебраического кодирования, разработанная в 1978 году Робертом Мак-Элисом. Это была первая схема, использующая рандомизацию в процессе шифрования. Алгоритм не получил широко признания в криптографии, но в то же время является кандидатом для постквантовой криптографии, так как устойчив к атаке с использованием Алгоритма Шора.

Алгоритм основан на сложности декодирования полных линейных кодов (общая задача декодирования является NP-сложной).

Блок-схема алгоритма представлена на рисунке 1.1.

Рисунок 1.2 – Блок-схемы алгоритма генерации ключей и шифрования сообщения Мак-Элиса

McEliece состоит из трех алгоритмов:

– алгоритма случайной генерации ключа, который дает открытый и секретный ключ;

– алгоритма случайного шифрования;

– детерминированного алгоритма расшифрования.

Текст сообщения представляет из себя вектор длины k над конечным полем GF(q).

Все пользователи в системе совместно используют параметры безопасности: n, k, t.

Генерация ключа описана на рисунке 1.2.

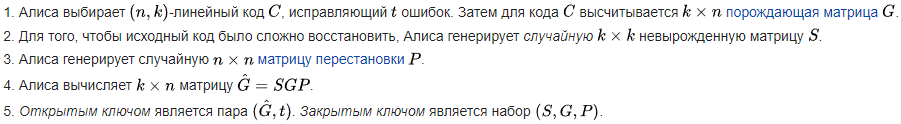


Рисунок 1.2 — Генерация ключа

Шифрование и дешифрование сообщения описаны на рисунках 1.3–1.4.

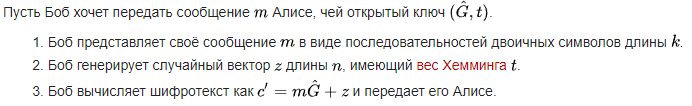


Рисунок 1.3 — Шифрование сообщения

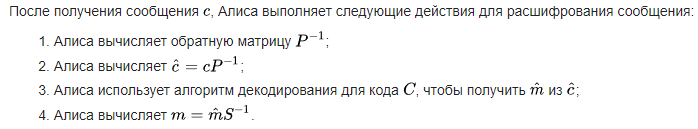


Рисунок 1.4 — Дешифрование сообщения

**2 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ**

Необходимо реализовать программные средства шифрования и дешифрования текстовых файлов при помощи алгоритма Мак-Элиса для криптостойких размеров порождающей матрицы. Программное средство реализовано на языке программирования Python. Демонстрация работы показана на рисунке 2.1

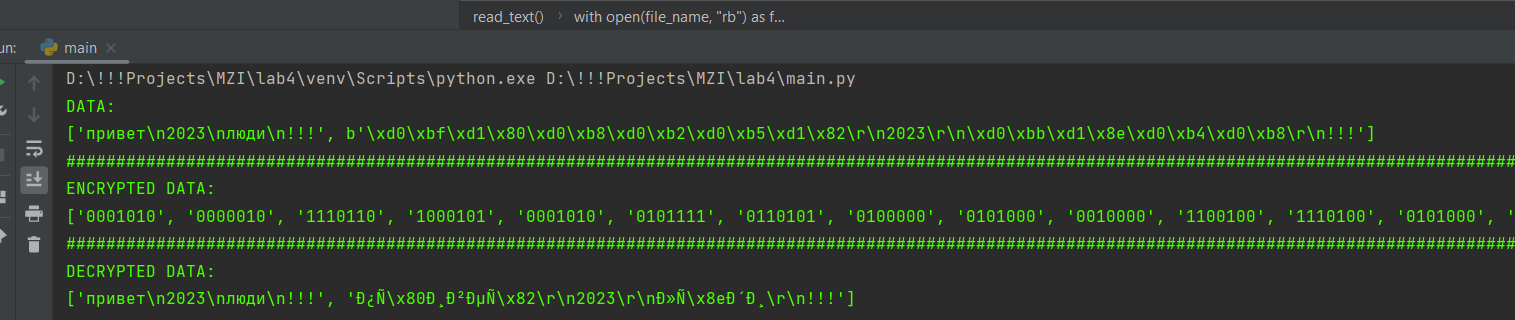


Рисунок 2.1 — Демонстрация работы

Вывод в консоль сделан для дополнительной наглядности, корректная работа с файлами показана на рисунке 2.2.

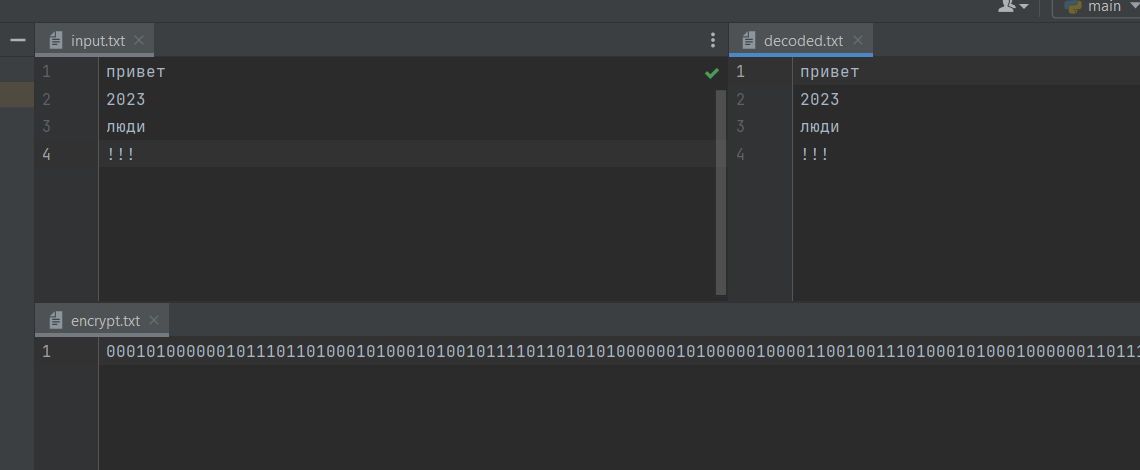


Рисунок 2.2 — Ввод/вывод файлов

# ЗАКЛЮЧЕНИЕ

Алгоритм Мак-Элиса - это криптографический алгоритм, основанный на комбинаторных проблемах, таких как задача о рюкзаке. Он обеспечивает теоретическую стойкость и широко применяется в области построения криптосистем с открытым ключом, которые сопряжены с комбинаторными сложностями. Однако, алгоритм Мак-Элиса имеет ограничения в эффективности и может потребовать больших вычислительных ресурсов. Он также может быть уязвим к атакам, таким как атаки на открытый текст. В целом, Мак-Элиса является интересным теоретическим алгоритмом, но обычно не используется в реальных приложениях криптографии в сравнении с более современными криптосистемами, такими как RSA и ECC.

# ПРИЛОЖЕНИЕ А

(**обязательное)**

**Листинг программного кода**

Файл main.py

import numpy as np

from mceliece import split\_binary\_string, hamming7\_4\_decode, add\_error, hamming7\_4\_encode, detect\_error, \

bits\_to\_str, flip\_bit, P\_inv, S\_inv

def read\_text(file\_name):

result = []

with open(file\_name, "r", encoding="utf-8") as file:

result.append(file.read())

with open(file\_name, "rb") as file:

result.append(file.read())

return result

def write\_text(file\_name, text):

with open(file\_name, "w", encoding="utf-8") as f:

f.write(text[0])

def encode(text):

binary\_str = ''.join(format(x, '08b') for x in text)

split\_bits\_list = split\_binary\_string(binary\_str, 4)

enc\_msg = []

for split\_bits in split\_bits\_list:

encode\_bits = hamming7\_4\_encode(split\_bits)

error\_bits = add\_error(encode\_bits)

str\_encode = ''.join(str(x) for x in error\_bits)

enc\_msg.append(str\_encode)

encoded = ''.join(enc\_msg)

with open("encrypt.txt", "w", encoding="utf-8") as f:

f.write(encoded)

return enc\_msg

def decode(encode\_msg):

decode\_msg = []

for encode\_bits in encode\_msg:

encode\_bits = np.array([int(x) for x in encode\_bits])

c\_hat = np.mod(encode\_bits.dot(P\_inv), 2)

error\_idx = detect\_error(c\_hat)

flip\_bit(c\_hat, error\_idx)

m\_hat = hamming7\_4\_decode(c\_hat)

m\_out = np.mod(m\_hat.dot(S\_inv), 2)

str\_dec = ''.join(str(x) for x in m\_out)

decode\_msg.append(str\_dec)

decode\_msg\_str = ''.join(decode\_msg)

return decode\_msg\_str

if \_\_name\_\_ == '\_\_main\_\_':

text = read\_text("input.txt")

print(f"DATA:\n{text}")

print('#' \* 150)

result = [text[0]]

encode\_msg = encode(text[1])

print(f"ENCRYPTED DATA:\n{encode\_msg}")

print('#' \* 150)

decode\_msg = decode(encode\_msg)

txt = bits\_to\_str(decode\_msg)

result.append(txt)

print(f"DECRYPTED DATA:\n{result} ")

write\_text("decoded.txt", result)