Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет   
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина «Методы защиты информации»

|  |  |
| --- | --- |
|  |  |

**ОТЧЕТ**

к лабораторной работе № 4

на тему «Асимметричная криптография. Алгоритм Мак-Элиса»

Выполнил             Д. С. Кончик

Проверил                          А. В. Герчик

Минск 2024

**СОДЕРЖАНИЕ**

[1 Цель лабораторной работы 3](#_Toc157722973)

[2 Теоретические сведения 4](#_Toc157722974)

[3 Результат выполнения 5](#_Toc157722975)

[Вывод](#_Toc157722976) 6

[Список использованных источников 7](#_Toc157722977)

[Приложение А (обязательное) Листинг исходного кода 8](#_Toc157722978)

# **1 ЦЕЛЬ ЛАБОРАТОРНОЙ РАБОТЫ**

Целью лабораторной работы является реализация программного средства шифрования и дешифрования текстовых файлов при помощи алгоритма Мак-Элиса. Также необходимо добавить интерфейс командной строки, который позволит выбирать файлы для шифрования/дешифрования.

# **2 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 основан на сложности декодирования полных линейных кодов.

Криптосистема имеет несколько преимуществ, например, над RSA. Шифрование и дешифрование проходит быстрее и с ростом длины ключа степень защиты данных растет гораздо быстрее. McEliece применим также в задачах аутентификации.

Идея, лежащая в основе данной системы, состоит в выборе корректирующего кода, исправляющего определенное число ошибок, для которого существует эффективный алгоритм декодирования. С помощью секретного ключа этот код «маскируется» под общий линейный код, для которого задача декодирования не имеет эффективного решения.

В системе Мак-Элиса параметрами системы, общими для всех абонентов, являются числа k, n, t. Для получения открытого и соответствующего секретного ключа каждому из абонентов системы следует осуществить следующие действия:

1 Выбрать порождающую матрицу G = Gkn двоичного (n,k)-линейного кода, исправляющего t ошибок, для которого известен эффективный алгоритм декодирования.

2 Случайно выбрать двоичную невырожденную матрицу S = Sk.

3 Случайно выбрать подстановочную матрицу P = Pn.

4 Вычислить произведение матриц G1 = S · G · P.

Открытым ключом является пара (G1, t), секретным – тройка (S, G, P).

Для того чтобы зашифровать сообщение M, предназначенное для абонента A, абоненту B следует выполнить следующие действия:

1 Представить M в виде двоичного вектора длины k.

2 Выбрать случайный бинарный вектор ошибок Z длиной n, содержащий не более t единиц.

3 Вычислить бинарный вектор C = M · GA + Z.

Получив сообщение C, абонент A вычисляет вектор C1 = C · P-1, с помощью которого, используя алгоритм декодирования кода с порождающей матрицей G, получает далее векторы M1 и M = M1 · S-1.

# **3 РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ**

В ходе выполнения лабораторной было реализовано программное средство шифрования и дешифрования текстовых файлов при помощи криптосистемы Мак-Элиса. Консольный интерфейс программы представлен на рисунке 1.

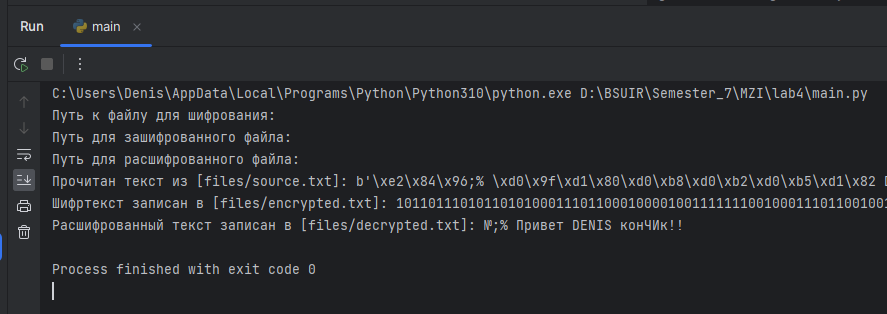


Рисунок 1 – Работа программы

На рисунках 2-4 представлены скриншоты файлов, с которыми работала программа.

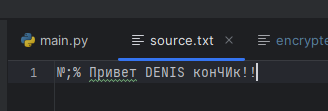


Рисунок 2 – Файл с открытым текстом

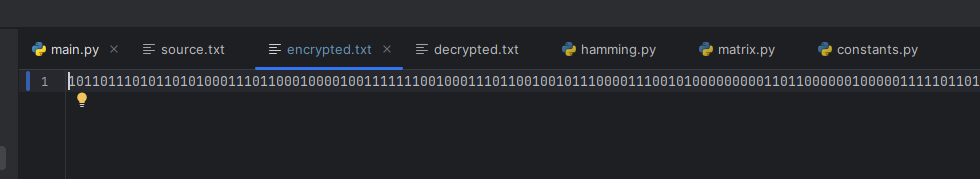


Рисунок 3 – Файл с шифртекстом

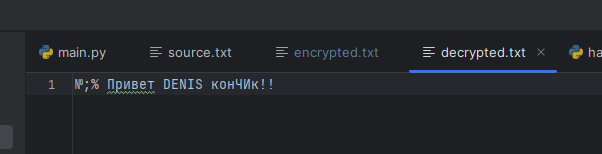


Рисунок 4 – Файл с расшифрованным текстом

# **ВЫВОД**

В ходе данной лабораторной работы было реализовано программное средство шифрования и дешифрования текстовых файлов при помощи криптосистемы Мак-Элиса. Программа реализована на языке программирования Python, предоставляет консольный интерфейс для выбора файла с открытым текстом и файлов для шифр- и расшифрованного текстов.

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

[1] McEliece [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/McEliece. – Дата доступа: 13.11.2024.

[2] Линейный код [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/Линейный\_код. – Дата доступа: 13.11.2024.

## **ПРИЛОЖЕНИЕ А**

## **(обязательное)**

## **Листинг исходного кода**

import numpy as np

# Проверочная матрица для вычисления синдрома при декодировании

H = np.array([[1, 0, 1, 0, 1, 0, 1],

[0, 1, 1, 0, 0, 1, 1],

[0, 0, 0, 1, 1, 1, 1]])

# Матрица преобразования 4-ых бит в 7

G = np.array([[1, 1, 0, 1],

[1, 0, 1, 1],

[1, 0, 0, 0],

[0, 1, 1, 1],

[0, 1, 0, 0],

[0, 0, 1, 0],

[0, 0, 0, 1]])

# Матрица преобразования 7-ми бит в 4

R = np.array([[0, 0, 1, 0, 0, 0, 0],

[0, 0, 0, 0, 1, 0, 0],

[0, 0, 0, 0, 0, 1, 0],

[0, 0, 0, 0, 0, 0, 1]])

def hamming7\_4\_encode(input\_str: str, G\_hat: np.ndarray) -> np.ndarray:

input\_bits = np.array([int(bit) for bit in input\_str])

encoded\_bits = np.mod(G\_hat.dot(input\_bits), 2)

return encoded\_bits

def hamming7\_4\_decode(encoded\_bits: np.ndarray) -> np.ndarray:

decoded\_bits = np.mod(R.dot(encoded\_bits), 2)

return decoded\_bits

def add\_single\_bit\_error(encoded\_bits: np.ndarray) -> np.ndarray:

error\_bits = [0] \* 7

error\_index = random.randint(0, 6)

error\_bits[error\_index] = 1

corrupted\_bits = np.mod(encoded\_bits + error\_bits, 2)

return corrupted\_bits

def detect\_error(received\_bits: np.ndarray) -> int:

syndrome = np.mod(H.dot(received\_bits), 2)

err\_index = int(''.join(str(bit) for bit in syndrome[::-1]), 2)

return err\_index - 1

def flip\_bit(bits: np.ndarray, index: int) -> None:

bits[index] = (bits[index] + 1) % 2

SOURCE\_PATH = 'files/source.txt'

ENCRYPTED\_PATH = 'files/encrypted.txt'

DECRYPTED\_PATH = 'files/decrypted.txt'

def generate\_non\_singular\_binary\_matrix(n: int) -> np.ndarray:

while True:

matrix: np.ndarray = np.random.randint(0, 2, size=(n, n))

if np.linalg.det(matrix) != 0:

return matrix

def generate\_random\_permutation\_matrix(n: int) -> np.ndarray:

perm\_indices = np.random.permutation(n)

permutation\_matrix = np.zeros((n, n), dtype=int)

for i in range(n):

permutation\_matrix[i, perm\_indices[i]] = 1

return permutation\_matrix

def split\_binary\_string(binary\_str: str, chunk\_size: int) -> list[str]:

return [binary\_str[i:i + chunk\_size] for i in range(0, len(binary\_str), chunk\_size)]

def bits\_to\_str(bits: str) -> str:

byte\_chunks = [bits[i:i + 8] for i in range(0, len(bits), 8)]

characters = [chr(int(chunk, 2)) for chunk in byte\_chunks]

return ''.join(characters)

def input\_with\_default(prompt: str, default\_value: str) -> str:

return input(prompt) or default\_value

def binary\_string\_to\_bytes(binary\_string: str) -> bytes:

byte\_chunks = [binary\_string[i:i + 8] for i in range(0, len(binary\_string), 8)]

byte\_array = bytearray(int(chunk, 2) for chunk in byte\_chunks)

return bytes(byte\_array)

# k x k невырожденная матрица

S: np.ndarray = generate\_non\_singular\_binary\_matrix(4)

S\_inv: np.ndarray = np.linalg.inv(S).astype(int)

# n x n матрица перестановки

P: np.ndarray = generate\_random\_permutation\_matrix(7)

P\_inv: np.ndarray = np.linalg.inv(P).astype(int)

G\_hat = np.transpose(np.mod((S.dot(np.transpose(G))).dot(P), 2))

def main() -> None:

source\_file\_path: str = input\_with\_default("Путь к файлу для шифрования: ", SOURCE\_PATH)

encrypted\_file\_path: str = input\_with\_default("Путь для зашифрованного файла: ", ENCRYPTED\_PATH)

decrypted\_file\_path: str = input\_with\_default("Путь для расшифрованного файла: ", DECRYPTED\_PATH)

with open(source\_file\_path, "rb") as file:

input\_bytes = file.read()

print(f"Прочитан текст из [{source\_file\_path}]: {input\_bytes}")

binary\_input: str = ''.join(f"{byte:08b}" for byte in input\_bytes)

binary\_blocks: list[str] = split\_binary\_string(binary\_input, 4)

encrypted\_blocks: list[str] = []

for block in binary\_blocks:

encoded\_block: np.ndarray = hamming7\_4\_encode(block, G\_hat)

corrupted\_block: np.ndarray = add\_single\_bit\_error(encoded\_block)

encoded\_str: str = ''.join(str(bit) for bit in corrupted\_block)

encrypted\_blocks.append(encoded\_str)

encrypted\_data = ''.join(encrypted\_blocks)

with open(encrypted\_file\_path, "w", encoding="utf-8") as file:

file.write(encrypted\_data)

print(f"Шифртекст записан в [{encrypted\_file\_path}]: {encrypted\_data}")

decoded\_messages: list[str] = []

for encrypted\_block in encrypted\_blocks:

encrypted\_bits: np.ndarray = np.array([int(bit) for bit in encrypted\_block])

c\_hat: np.ndarray = np.mod(encrypted\_bits.dot(P\_inv), 2)

error\_index: int = detect\_error(c\_hat)

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

m\_hat: np.ndarray = hamming7\_4\_decode(c\_hat)

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

decoded\_str: str = ''.join(str(bit) for bit in m)

decoded\_messages.append(decoded\_str)

decoded\_data: str = ''.join(decoded\_messages)

decoded\_text: str = binary\_string\_to\_bytes(decoded\_data).decode('utf-8')

with open(decrypted\_file\_path, "w", encoding="utf-8", newline='') as file:

file.write(decoded\_text)

print(f"Расшифрованный текст записан в [{decrypted\_file\_path}]: {decoded\_text}")

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

main()