

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
2 Теоретические сведения	4
3 Результат выполнения	4
Вывод.....	6
Список использованных источников	7
Приложение А (обязательное) Листинг исходного кода	8

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 $G = G_{kn}$ двоичного (n,k) -линейного кода, исправляющего t ошибок, для которого известен эффективный алгоритм декодирования.

- 2 Случайно выбрать двоичную невырожденную матрицу $S = S_k$.

- 3 Случайно выбрать подстановочную матрицу $P = P_n$.

- 4 Вычислить произведение матриц $G_1 = S \cdot G \cdot P$.

Открытым ключом является пара (G_1, t) , секретным – тройка (S, G, P) .

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

- 1 Представить M в виде двоичного вектора длины k .

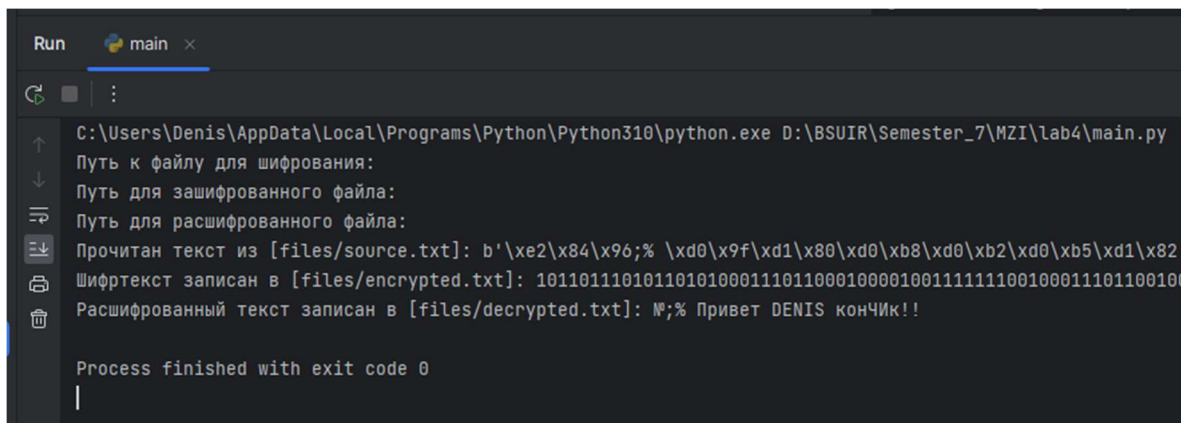
- 2 Выбрать случайный бинарный вектор ошибок Z длиной n , содержащий не более t единиц.

- 3 Вычислить бинарный вектор $C = M \cdot G_1 + Z$.

Получив сообщение C , абонент А вычисляет вектор $C_1 = C \cdot P^{-1}$, с помощью которого, используя алгоритм декодирования кода с порождающей матрицей G , получает далее векторы M_1 и $M = M_1 \cdot S^{-1}$.

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.

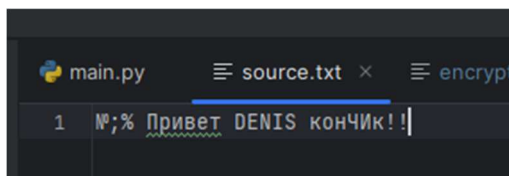


```
Run main x
C:\Users\Denis\AppData\Local\Programs\Python\Python310\python.exe D:\BSUIR\Semester_7\МЗИ\lab4\main.py
Путь к файлу для шифрования:
Путь для зашифрованного файла:
Путь для расшифрованного файла:
Прочитан текст из [files/source.txt]: b'\xe2\x84\x96;% \xd0\x9f\xd1\x80\xd0\xb8\xd0\xb2\xd0\xb5\xd1\x82'
Шифртекст записан в [files/encrypted.txt]: 1011011101011010100011101100010000100111111100100011101100100
Расшифрованный текст записан в [files/decrypted.txt]: №;% Привет DENIS кончик!!

Process finished with exit code 0
```

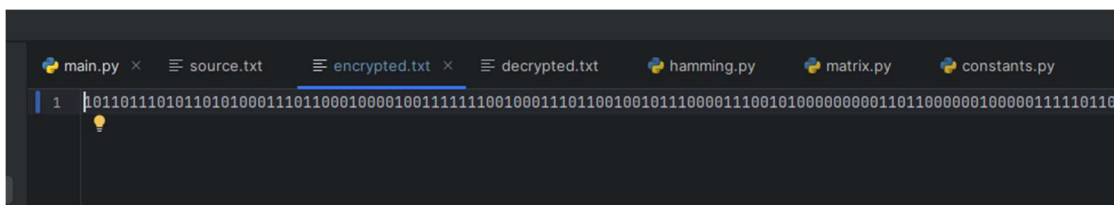
Рисунок 1 – Работа программы

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



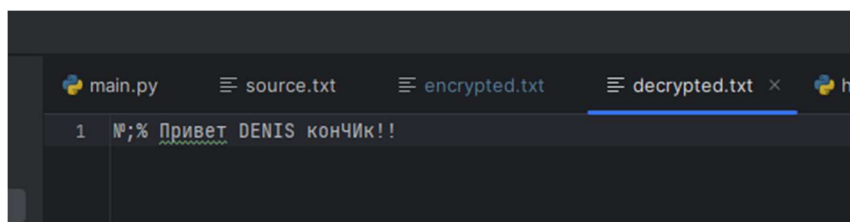
```
main.py source.txt encrypted.txt
1 №;% Привет DENIS кончик!!
```

Рисунок 2 – Файл с открытым текстом



```
main.py source.txt encrypted.txt decrypted.txt hamming.py matrix.py constants.py
1 10110111010110101000111011000100001001111111001000111011001001011100001110010100000000110110000001000001111101101
```

Рисунок 3 – Файл с шифртекстом



```
main.py source.txt encrypted.txt decrypted.txt hamming.py
1 №;% Привет DENIS кончик!!
```

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()

```