Міністерство освіти і науки України
Дніпровський національний університет імені Олеся Гончара
Факультет прикладної математики і комп’ютерних технологій
Кафедра комп’ютерних технологій

ЗВІТ З ЛАБОРАТОРНОЇ РОБОТИ № 3

з курсу «Методи кібербезпеки»

на тему «Алгоритм шифрування AES»

Варіант №14

Виконав:
студент гр. ПА-22-2
Овдієнко Андрій

Дніпро
2025

Зміст

1. Постановка задачі

Програмно реалізувати алгоритми шифрування AES.

2. Опис розв'язку

Винайдене у Національному Інституті Стандатртів та Технологій (NIST), за Федеральним Стандартом Обробки Інформації (FIPS PUB 197) (1997 року), метод шифрування Advanced Encryption Standard (AES) надає надійний криптографічний алгоритм для захисту електронних даних.

AES-алгоритм це симетричний блоковий шифр. Шифрування конвертує дані (впорядкований текст) до нечитаної форми, що називається зашифрованим текстом, а процес розшифрування конвертує дані у зворотному порядку. Криптографічний ключ має довжину 128, 192 або 256 біт (що впливає довжину ключа та кількість раундів), що дозволяє зберігати дані розбитими на зашифровані блоки по 128 біт.

Можна представити що перебір всіх можливих комбінацій:

- AES – 128: 2^{128};
- AES – 192: 2^{192};
- AES – 256: 2^{256}.

У цій роботі буде зроблено алгоритм AES-128.

128 біт – це 16 байт, тобто будемо працювати з матрицею 4 на 4. Буде 11 ключів. 10 раундів.

Щоб отримати 11 різних ключів із 16 байт – слід розуміти, що перший ключ дається користуввачем 16 байт, а інші робляться за допомогою попереднього масиву, Rcon та S-box.

Нульовий ключ заповнюється наступним чином. Нехай в нас є 16 байтів (16 елементів), і скажімо, що вони маються назви $\{k_0 \quad k_1 \quad \cdots \quad k_{15}\}$. Тоді заповнення нульової матриці таке:

$$\begin{bmatrix} k_0 & k_1 & k_2 & k_3 \\ k_4 & k_5 & k_6 & k_7 \\ k_8 & k_9 & k_{10} & k_{11} \\ k_{12} & k_{13} & k_{14} & k_{15} \end{bmatrix}$$

Алгоритм створення нового ключа такий:

- Береться останній стовпець попереднього ключа.

- Відбувається зсув елементів, верхній елемент йде на низ – а всі інші крокують вгору на 1 елемент.
- Відбувається заміна у цьому стовпці кожного елемента на відповідний елемент у матриці S-box. Наприклад, був елемент “cf” – шукаємо рядок з “c”, шукаємо стовпець із “f” – це буде “8a”. нагадаю, що всевідбувається у шістнадцятковій системі. Щоб себе перевірити – можете зробити заміну для елементів {4f; 3c; 09} – це буде (після заміни з таблиці S-box) {84; eb; 01}.
- Береться перший стовпець попередньої матриці ключів, робиться операція XOR зі стовпцем, який ми отримали на попередньому кроці таробиться операція XOR з відповідним стовпцем у матриці Rcon (стовпець обирається наступним чином (для першого ключа і до десятого, бо в нас вже є нульовий – це наш ключ, який ми отримали від користувача) – спочатку стовпець $\begin{matrix}01\\00\\00\\00\end{matrix}$, потім $\begin{matrix}02\\00\\00\\00\end{matrix}$ і так далі). Отримали перший стовпець новго ключа.
- Щоб отримати наступні 3 стовпці слід зробити операцію XOR двох столбців. Наприклад ми шукаємо другий стовпець нового ключа – нам слід зробити операцію XOR між другим стовпцем попереднього ключа і першим стовпцем новго ключа. Якщо ми шукаємо третій стовпець нового ключа – нам слід зробити операцію XOR між третім стовпцем попереднього ключа і другим стовпцем новго ключа. Якщо ми шукаємо четвертий стовпець нового ключа – нам слід зробити операцію XOR між четвертим стовпцем попереднього ключа і третім стовпцем новго ключа.
- У AES-128 слід зробити поточну операцію 10 разів, щоб в сумі отримати 11 ключів.

Зобразимо алгоритм AES у вигляді блок-схеми для одного блоку (16 байт) (Рисунок 1).

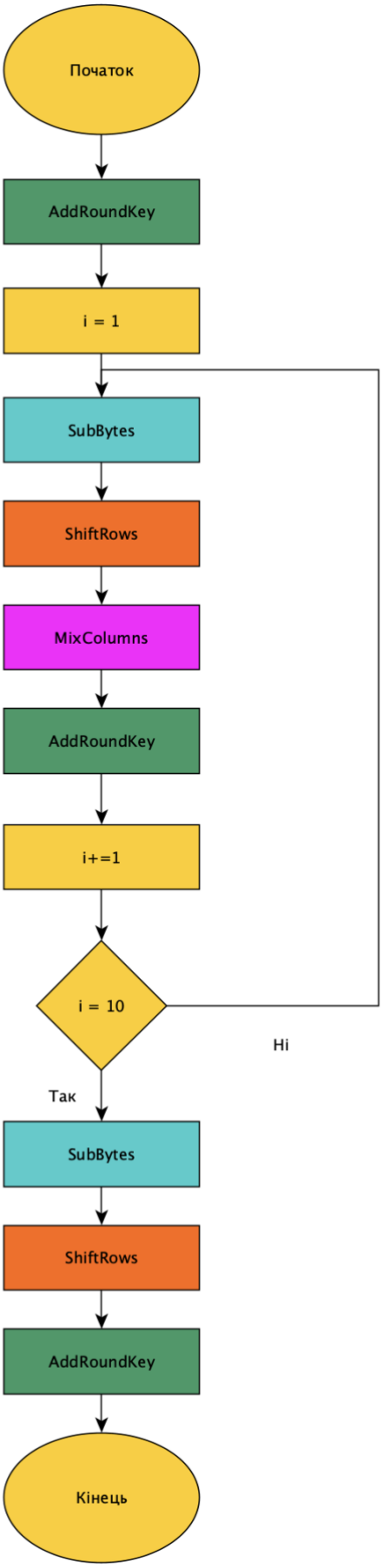

Рисунок 1 – Блок-схема ітерації для блоку 16 байт.

Один блок 16 байт матриці state заповнюється наступним чином.

Нехай в нас є 16 байтів (16 елементів), і скажімо, що вони маються назви $\{s_0 \quad s_1 \quad \cdots \quad s_{15}\}$. Тоді заповнення матриці state таке:

$$\begin{bmatrix} s_0 & s_4 & s_8 & s_{12} \\ s_1 & s_5 & s_9 & s_{13} \\ s_2 & s_6 & s_{10} & s_{14} \\ s_3 & s_7 & s_{11} & s_{15} \end{bmatrix}$$

AddRoundKey

Це процес XOR між матрицею state та від повідним ключом key.

$$\begin{bmatrix} s_0 & s_4 & s_8 & s_{12} \\ s_1 & s_5 & s_9 & s_{13} \\ s_2 & s_6 & s_{10} & s_{14} \\ s_3 & s_7 & s_{11} & s_{15} \end{bmatrix} \oplus \begin{bmatrix} k_0 & k_1 & k_2 & k_3 \\ k_4 & k_5 & k_6 & k_7 \\ k_8 & k_9 & k_{10} & k_{11} \\ k_{12} & k_{13} & k_{14} & k_{15} \end{bmatrix} = \begin{bmatrix} s_0 \oplus k_0 & s_4 \oplus k_1 & s_8 \oplus k_2 & s_{12} \oplus k_3 \\ s_1 \oplus k_4 & s_5 \oplus k_5 & s_9 \oplus k_6 & s_{13} \oplus k_7 \\ s_2 \oplus k_8 & s_6 \oplus k_9 & s_{10} \oplus k_{10} & s_{14} \oplus k_{11} \\ s_3 \oplus k_{12} & s_7 \oplus k_{13} & s_{11} \oplus k_{14} & s_{15} \oplus k_{15} \end{bmatrix}$$

При розшифруванні просто матриці ключів йдут від останнього до нульового.

SubBytes

Це процес заміни у шістнадцятирічній системі елементів матриці State на елементи в матриці S-box. Для прикладу візьмемо матрицю State у шістнадцятирічній системі.

$$\begin{bmatrix} 00 & 06 & 0c & 0f \\ 72 & 17 & 38 & d5 \\ c4 & d6 & d4 & e5 \\ f3 & f6 & 73 & ff \end{bmatrix} \rightarrow S - box \rightarrow \begin{bmatrix} 63 & 6f & fe & 76 \\ 40 & f0 & 07 & 03 \\ 1c & f6 & 48 & d9 \\ 0d & 42 & 8f & 16 \end{bmatrix}$$

У розшифруванні використовується замість S-box обернена їй матриця.

SiftRows

Процес зсуву state. Перший рядок залишається на місці. Другий рядок на 1 елемент ліворуч, третій на 2, четвертий на 3.

$$\begin{bmatrix} s_0 & s_4 & s_8 & s_{12} \\ s_1 & s_5 & s_9 & s_{13} \\ s_2 & s_6 & s_{10} & s_{14} \\ s_3 & s_7 & s_{11} & s_{15} \end{bmatrix} \rightarrow ShifrRows \rightarrow \begin{bmatrix} s_0 & s_4 & s_8 & s_{12} \\ s_5 & s_9 & s_{13} & s_1 \\ s_{10} & s_{14} & s_2 & s_6 \\ s_{15} & s_3 & s_7 & s_{11} \end{bmatrix}$$

У розшифруванні робиться аналогічна операція, але зсув праворуч.

MixColumns

Множення матриці на кожень стовпець state. Але замість плюса – операція XOR. А замість множення - множення з використанням полів Галуа. Матриця переходу:

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix}$$

А множення відбувається наступним чином. Уявімо два елемента a та b.

Представимо у двійковій системі. Наприклад a = 8; b = 9, result = 0.

8 = 0000 1000;

9 = 0000 1001;

result = 0000 0000;

перевіряємо молодший біт 9 – це 1 – тому result = 0000 0000 XOR 0000 1000 = 0000 1000; якщо молодший біт 0 – нічого не робимо;

перевіряємо старший біт 8 – це 0, при здвигу ліворуч – нічого не вилізе;

a зсуваємо вліво, b вправо;

a = 0001 0000;

b = 0000 01000;

result = 0000 1000;

так зробити 8 разів і результат повернути.

Тепер увага, при випадку, коли старший біт числа a = 1.

a = 1000 0000;

b = 0000 0010;

молодший біт 9 – 0, нічого не робимо;

старший біт a = 1;

a зсуваємо вліво, b вправо;

a = 0000 0000;

b = 0000 00001;

так як старший біт БУВ 1 у a – то a = a XOR 0001 1011 = 0001 1011.

Тобто якщо старший біт a = 1 – то потім відбувається a = a XOR 0001 1011.

Для розшифрування:

$$\begin{bmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{bmatrix}$$

Важливий момент

У розшифруванні міняється місцями операції MixColumn та AddRoundKey. Але всеодно MixColumn не робиться на кінцевому етапі, а AddRoundKey робиться на кожному.

3. Код програми

```
import tkinter as tk
from tkinter import font, ttk, filedialog, messagebox, scrolledtext

CONST_WIDTH_TEXT = 40
CONST_HEIGHT_TEXT = 7

text = ""
key = ""
filename = ""

s_box = [
    0x63, 0x7C, 0x77, 0x7B, 0xF2, 0x6B, 0x6F, 0xC5,
    0x30, 0x01, 0x67, 0x2B, 0xFE, 0xD7, 0xAB, 0x76,
    0xCA, 0x82, 0xC9, 0x7D, 0xFA, 0x59, 0x47, 0xF0,
    0xAD, 0xD4, 0xA2, 0xAF, 0x9C, 0xA4, 0x72, 0xC0,
    0xB7, 0xFD, 0x93, 0x26, 0x36, 0x3F, 0xF7, 0xCC,
    0x34, 0xA5, 0xE5, 0xF1, 0x71, 0xD8, 0x31, 0x15,
    0x04, 0xC7, 0x23, 0xC3, 0x18, 0x96, 0x05, 0x9A,
    0x07, 0x12, 0x80, 0xE2, 0xEB, 0x27, 0xB2, 0x75,
    0x09, 0x83, 0x2C, 0x1A, 0x1B, 0x6E, 0x5A, 0xA0,
    0x52, 0x3B, 0xD6, 0xB3, 0x29, 0xE3, 0x2F, 0x84,
    0x53, 0xD1, 0x00, 0xED, 0x20, 0xFC, 0xB1, 0x5B,
    0x6A, 0xCB, 0xBE, 0x39, 0x4A, 0x4C, 0x58, 0xCF,
    0xD0, 0xEF, 0xAA, 0xFB, 0x43, 0x4D, 0x33, 0x85,
    0x45, 0xF9, 0x02, 0x7F, 0x50, 0x3C, 0x9F, 0xA8,
    0x51, 0xA3, 0x40, 0x8F, 0x92, 0x9D, 0x38, 0xF5,
    0xBC, 0xB6, 0xDA, 0x21, 0x10, 0xFF, 0xF3, 0xD2,
    0xCD, 0x0C, 0x13, 0xEC, 0x5F, 0x97, 0x44, 0x17,
    0xC4, 0xA7, 0x7E, 0x3D, 0x64, 0x5D, 0x19, 0x73,
    0x60, 0x81, 0x4F, 0xDC, 0x22, 0x2A, 0x90, 0x88,
    0x46, 0xEE, 0xB8, 0x14, 0xDE, 0x5E, 0x0B, 0xDB,
    0xE0, 0x32, 0x3A, 0x0A, 0x49, 0x06, 0x24, 0x5C,
    0xC2, 0xD3, 0xAC, 0x62, 0x91, 0x95, 0xE4, 0x79,
```

```
        0xE7, 0xC8, 0x37, 0x6D, 0x8D, 0xD5, 0x4E, 0xA9,
        0x6C, 0x56, 0xF4, 0xEA, 0x65, 0x7A, 0xAE, 0x08,
        0xBA, 0x78, 0x25, 0x2E, 0x1C, 0xA6, 0xB4, 0xC6,
        0xE8, 0xDD, 0x74, 0x1F, 0x4B, 0xBD, 0x8B, 0x8A,
        0x70, 0x3E, 0xB5, 0x66, 0x48, 0x03, 0xF6, 0x0E,
        0x61, 0x35, 0x57, 0xB9, 0x86, 0xC1, 0x1D, 0x9E,
        0xE1, 0xF8, 0x98, 0x11, 0x69, 0xD9, 0x8E, 0x94,
        0x9B, 0x1E, 0x87, 0xE9, 0xCE, 0x55, 0x28, 0xDF,
        0x8C, 0xA1, 0x89, 0x0D, 0xBF, 0xE6, 0x42, 0x68,
        0x41, 0x99, 0x2D, 0x0F, 0xB0, 0x54, 0xBB, 0x16
    ]

    r_con = [
        [0x01, 0x00, 0x00, 0x00],
        [0x02, 0x00, 0x00, 0x00],
        [0x04, 0x00, 0x00, 0x00],
        [0x08, 0x00, 0x00, 0x00],
        [0x10, 0x00, 0x00, 0x00],
        [0x20, 0x00, 0x00, 0x00],
        [0x40, 0x00, 0x00, 0x00],
        [0x80, 0x00, 0x00, 0x00],
        [0x1B, 0x00, 0x00, 0x00],
        [0x36, 0x00, 0x00, 0x00]
    ]

    xob_s = [
        0x52, 0x09, 0x6A, 0xD5, 0x30, 0x36, 0xA5, 0x38, 0xBF, 0x40, 0xA3, 0x9E, 0x81, 0xF3,
0xD7, 0xFB,
        0x7C, 0xE3, 0x39, 0x82, 0x9B, 0x2F, 0xFF, 0x87, 0x34, 0x8E, 0x43, 0x44, 0xC4, 0xDE,
0xE9, 0xCB,
        0x54, 0x7B, 0x94, 0x32, 0xA6, 0xC2, 0x23, 0x3D, 0xEE, 0x4C, 0x95, 0x0B, 0x42, 0xFA,
0xC3, 0x4E,
        0x08, 0x2E, 0xA1, 0x66, 0x28, 0xD9, 0x24, 0xB2, 0x76, 0x5B, 0xA2, 0x49, 0x6D, 0x8B,
0xD1, 0x25,
```

```
            0x72, 0xF8, 0xF6, 0x64, 0x86, 0x68, 0x98, 0x16, 0xD4, 0xA4, 0x5C, 0xCC, 0x5D, 0x65,
0xB6, 0x92,
            0x6C, 0x70, 0x48, 0x50, 0xFD, 0xED, 0xB9, 0xDA, 0x5E, 0x15, 0x46, 0x57, 0xA7, 0x8D,
0x9D, 0x84,
            0x90, 0xD8, 0xAB, 0x00, 0x8C, 0xBC, 0xD3, 0x0A, 0xF7, 0xE4, 0x58, 0x05, 0xB8, 0xB3,
0x45, 0x06,
            0xD0, 0x2C, 0x1E, 0x8F, 0xCA, 0x3F, 0x0F, 0x02, 0xC1, 0xAF, 0xBD, 0x03, 0x01, 0x13,
0x8A, 0x6B,
            0x3A, 0x91, 0x11, 0x41, 0x4F, 0x67, 0xDC, 0xEA, 0x97, 0xF2, 0xCF, 0xCE, 0xF0, 0xB4,
0xE6, 0x73,
            0x96, 0xAC, 0x74, 0x22, 0xE7, 0xAD, 0x35, 0x85, 0xE2, 0xF9, 0x37, 0xE8, 0x1C, 0x75,
0xDF, 0x6E,
            0x47, 0xF1, 0x1A, 0x71, 0x1D, 0x29, 0xC5, 0x89, 0x6F, 0xB7, 0x62, 0x0E, 0xAA, 0x18,
0xBE, 0x1B,
            0xFC, 0x56, 0x3E, 0x4B, 0xC6, 0xD2, 0x79, 0x20, 0x9A, 0xDB, 0xC0, 0xFE, 0x78, 0xCD,
0x5A, 0xF4,
            0x1F, 0xDD, 0xA8, 0x33, 0x88, 0x07, 0xC7, 0x31, 0xB1, 0x12, 0x10, 0x59, 0x27, 0x80,
0xEC, 0x5F,
            0x60, 0x51, 0x7F, 0xA9, 0x19, 0xB5, 0x4A, 0x0D, 0x2D, 0xE5, 0x7A, 0x9F, 0x93, 0xC9,
0x9C, 0xEF,
            0xA0, 0xE0, 0x3B, 0x4D, 0xAE, 0x2A, 0xF5, 0xB0, 0xC8, 0xEB, 0xBB, 0x3C, 0x83,
0x53, 0x99, 0x61,
            0x17, 0x2B, 0x04, 0x7E, 0xBA, 0x77, 0xD6, 0x26, 0xE1, 0x69, 0x14, 0x63, 0x55, 0x21,
0x0C, 0x7D
        ]

        def add_empty_to_16(value):
            temp = value
            value = ""

            i = 0
            while len(value) < len(temp) or len(value) % 16 != 0:
                if i >= len(temp):
                    value += '\0'
                else:
```

```
            value += temp[i]
        i += 1

    return value

def edit_16_elements_in_one(value):
    result = []
    temp = []

    for i,char in enumerate(value):
        temp.append(char)

        if i % 16 == 0 and i != 0:
            result.append(temp)
            temp = []

    if temp:
        result.append(temp)

    return result

'''
Для AES.
'''
def mult(a, b):

    result = 0
    for _ in range(8):
        if b & 1:
            result ^= a
        hi_bit_set = a & 0x80
        a <<= 1
        if hi_bit_set:
            a ^= 0x1B
        b >>= 1
```

```
    return result % 256

def foo(text,key):

    '''
    Якщо чогось немає - виходимо.
    '''

    if not text or not key:
        print("Error: text or key is empty.")
        return ""

    '''
    Округляємо до 16 картності байтів.
    '''
    text = add_empty_to_16(text)
    key = add_empty_to_16(key)[:16] # Беремо перші 16 символів.

    '''
    Робимо ASCII
    '''
    text = [ord(char) for char in text]
    key = [ord(char) for char in key]


    '''
    Робимо масив масивів по 16 байт.
    '''
    temp = text
    text = []
    i=0
    while i<len(temp):
        demo = []
        for j in range(16):
            demo.append(temp[i+j])
```

```
            text.append(demo)
            i+=16

        '''
        Створюємо тип для кожного блоку state, temp_key
        '''
        s = [[None,None,None,None],[None,None,None,None],[None,None,None,None],[None,None,None,None]]
        k = [[None,None,None,None],[None,None,None,None],[None,None,None,None],[None,None,None,None]]

        '''
        Заповнюємо перший ключ.
        '''
        for i in range(4):
            for j in range(4):
                k[j][i] = int(key[i*4+j])
        key = [k]


        '''
        Заповнюємо інші 10 ключів.
        '''
        for i in range(10):

            '''
            Обнуляємо новий ключ.
            '''
            k = [[None, None, None, None], [None, None, None, None], [None, None, None, None], [None, None, None, None]]

            '''
            Останній стовпець останнього повного ключа.
```

```
            '''
            demo = []
            for j in range(4):
                demo.append(key[i][j][3])

            '''
            Зсув верхнього на низ і всі інші вгору на одну та беремо елементи з S-BOX.
            '''
            demo[0], demo[1], demo[2], demo[3] = s_box[demo[1]], s_box[demo[2]],
s_box[demo[3]], s_box[demo[0]]

            '''
            Перший стовпець нового ключа дорівнює
                першому стовпцю першого ключа
                ксор з
                Отриманим останнім стовпцем з сувом і заміною S-Box
                ксор з
                відповідним стовпцем матриці Rcon.
            '''
            for j in range(4):
                k[j][0] = key[i][j][0] ^ demo[j] ^ r_con[i][j]



            '''
            Останні стовпці новго ключа:
                відповідний стовпець прошлого ключа
                ксор з
                останнім поточно заповненим стовпцем нового ключа
            '''
            for n in range(1,4):
                demo = []
                for j in range(4):
                    demo.append(key[i][j][n])

                for j in range(4):
```

```
            k[j][n] = demo[j] ^ k[j][n - 1]

    key.append(k) # Додаємо новий ключ.



result = "" # строковийрезультат при виході.

'''
Обробка по 16 байтів.
'''
for _ in text:

    '''
    Заповнення поточної матриці state.
    По стовпцям.
    '''
    j = 0
    i = 0
    while i+3 < len(_) and j < len(s[0]):

        s[0][j] = _[i]
        s[1][j] = _[i + 1]
        s[2][j] = _[i + 2]
        s[3][j] = _[i + 3]

        i+=4
        j+=1

    '''
    Ксор матриці state з першим ключем.
    '''
    for i in range(4):
        for j in range(4):
            s[i][j] ^= key[0][i][j]
```

```
'''
Останні 10 райндів.
'''
for index in range(1,11):

    '''
    Заміна матрицею S-Box.
    '''
    for i, _ in enumerate(s):
        for j, _ in enumerate(s[i]):
            s[i][j] = s_box[s[i][j]]

    '''
    Зсув
    нульовий рядок не чіпаємо.
    перший рядок на 1 вліво.
    другий рядок на 2 вліво.
    третій рядок на 3 вліво.
    '''
    s[1][0], s[1][1], s[1][2], s[1][3] = s[1][1], s[1][2], s[1][3], s[1][0]

    s[2][0], s[2][1], s[2][2], s[2][3] = s[2][2], s[2][3], s[2][0], s[2][1]

    s[3][0], s[3][1], s[3][2], s[3][3] = s[3][3], s[3][0], s[3][1], s[3][2]


    '''
    Якщо не останній раунд.
    '''
    if index != 10:

        '''
        MixColumns
        '''
```

```
            for j in range(4):
                temp = [s[0][j], s[1][j], s[2][j], s[3][j]]

                s[0][j] = mult(0x02, temp[0]) ^ mult(0x03, temp[1]) ^ temp[2] ^ temp[3]
                s[1][j] = temp[0] ^ mult(0x02, temp[1]) ^ mult(0x03, temp[2]) ^ temp[3]
                s[2][j] = temp[0] ^ temp[1] ^ mult(0x02, temp[2]) ^ mult(0x03, temp[3])
                s[3][j] = mult(0x03, temp[0]) ^ temp[1] ^ temp[2] ^ mult(0x02, temp[3])


        '''
        Ксор state з відповідним ключем.
        '''
        for i in range(4):
            for j in range(4):
                s[j][i] = s[j][i]^key[index][j][i]

    '''
    Перенос матриці state до строкового типу у відповідь.
    '''
    demo = ""
    for i in range(4):
        for j in range(4):
            demo += chr(s[j][i])
    result += demo

  return result

def foo2(text, key):

  '''
  Якщо чогось немає - виходимо.
  '''

  if not text or not key:
    print("Error: text or key is empty.")
```

```
        return ""

    '''
    Округляємо до 16 картності байтів.
    '''
    text = add_empty_to_16(text)
    key = add_empty_to_16(key)[:16]  # Беремо перші 16 символів.

    '''
    Робимо ASCII
    '''
    text = [ord(char) for char in text]
    key = [ord(char) for char in key]

    '''
    Робимо масив масивів по 16 байт.
    '''
    temp = text
    text = []
    i = 0
    while i < len(temp):
        demo = []
        for j in range(16):
            demo.append(temp[i + j])
        text.append(demo)
        i += 16

    '''
    Створюємо тип для кожного блоку state, temp_key
    '''
    s = [[None, None, None, None], [None, None, None, None], [None, None, None, None],
[None, None, None, None]]
    k = [[None, None, None, None], [None, None, None, None], [None, None, None, None],
[None, None, None, None]]
```

```
        '''
        Заповнюємо перший ключ.
        '''
        for i in range(4):
            for j in range(4):
                k[j][i] = int(key[i * 4 + j])
        key = [k]

        '''
        Заповнюємо інші 10 ключів.
        '''
        for i in range(10):

            '''
            Обнуляємо новий ключ.
            '''
            k = [[None, None, None, None], [None, None, None, None], [None, None, None, None],
[None, None, None, None]]

            '''
            Останній стовпець останнього повного ключа.
            '''
            demo = []
            for j in range(4):
                demo.append(key[i][j][3])

            '''
            Зсув верхнього на низ і всі інші вгору на одну та беремо елементи з S-BOX.
            '''
            demo[0], demo[1], demo[2], demo[3] = s_box[demo[1]], s_box[demo[2]],
s_box[demo[3]], s_box[demo[0]]

            '''
            Перший стовпець нового ключа дорівнює
                першому стовпцю першого ключа
```

```
        ксор з
        Отриманим останнім стовпцем з сувом і заміною S-Box
        ксор з
        відповідним стовпцем матриці Rcon.
    '''
    for j in range(4):
        k[j][0] = key[i][j][0] ^ demo[j] ^ r_con[i][j]



    '''
    Останні стовпці новго ключа:
        відповідний стовпець прошлого ключа
        ксор з
        останнім поточно заповненим стовпцем нового ключа
    '''
    for n in range(1,4):
        demo = []
        for j in range(4):
            demo.append(key[i][j][n])

        for j in range(4):
            k[j][n] = demo[j] ^ k[j][n - 1]

    key.append(k) # Додаємо новий ключ.



result = ""  # строковийрезультат при виході.

'''
Обробка по 16 байтів.
'''
for _ in text:

    '''
    Заповнення поточної матриці state.
```

```
По стовпцям.
'''
j = 0
i = 0
while i + 3 < len(_) and j < len(s[0]):
    s[0][j] = _[i]
    s[1][j] = _[i + 1]
    s[2][j] = _[i + 2]
    s[3][j] = _[i + 3]

    i += 4
    j += 1

'''
Ксор матриці state з останнім ключем.
'''
for i in range(4):
    for j in range(4):
        s[i][j] ^= key[10][i][j]

for index in range(10):

    '''
    Заміна матрицею S-Box (обернена).
    '''
    for i, _ in enumerate(s):
        for j, _ in enumerate(s[i]):
            s[i][j] = xob_s[s[i][j]]

    '''
    Зсув
    нульовий рядок не чіпаємо.
    перший рядок на 1 вправо.
    другий рядок на 2 вправо.
    третій рядок на 3 вправо.
```

```
        '''
        s[1][0], s[1][1], s[1][2], s[1][3] = s[1][3], s[1][0], s[1][1], s[1][2]
        s[2][0], s[2][1], s[2][2], s[2][3] = s[2][2], s[2][3], s[2][0], s[2][1]
        s[3][0], s[3][1], s[3][2], s[3][3] = s[3][1], s[3][2], s[3][3], s[3][0]

        '''
        Ксор state з відповідним ключем (зворотний порядок).
        '''
        for i in range(4):
            for j in range(4):
                s[j][i] = s[j][i] ^ key[len(key) - 2 - index][j][i]

        '''
        MixColumns
        '''
        if index != 9:
            for j in range(4):
                col = [s[0][j], s[1][j], s[2][j], s[3][j]]
                s[0][j] = mult(0x0E, col[0]) ^ mult(0x0B, col[1]) ^ mult(0x0D,col[2]) ^ mult(0x09, col[3])
                s[1][j] = mult(0x09, col[0]) ^ mult(0x0E, col[1]) ^ mult(0x0B,col[2]) ^ mult(0x0D, col[3])
                s[2][j] = mult(0x0D, col[0]) ^ mult(0x09, col[1]) ^ mult(0x0E,col[2]) ^ mult(0x0B, col[3])
                s[3][j] = mult(0x0B, col[0]) ^ mult(0x0D, col[1]) ^ mult(0x09, col[2]) ^ mult(0x0E, col[3])


        '''
        Перенос матриці state до строкового типу у відповідь.
        '''
        demo = ""
        for i in range(4):
            for j in range(4):
```

```
                demo += chr(s[j][i])
            result += demo

        while True:
            if result[len(result)-1] != '\0':
                break
            result = result[:len(result)-1]

        return result

    def main(root):
        try:
            [widget.destroy() for widget in root.winfo_children()]
        except Exception:
            pass

        def input_text():

            def type(root, choice):

                def save(temp):
                    global text
                    text = temp
                    main(root)

                try:
                    [widget.destroy() for widget in root.winfo_children()]
                except Exception:
                    pass
                root.title("Input the text")
                button_back = tk.Button(root, text='<-', font=main_font, command=lambda:
input_text())
                button_back.place(x=10, y=10)

                root.update()
```

```
        if choice == "Write.":
            label = tk.Label(root, text="Enter your text:", font=main_font)
            label.place(x=0, y=0)

            entry = tk.Text(root, width=CONST_WIDTH_TEXT, height=CONST_HEIGHT_TEXT, font=main_font)
            entry.place(x=0, y=0)

            button = tk.Button(root, text='Save', font=main_font, command=lambda: save(entry.get("1.0", tk.END)))
            button.place(x=0, y=0)

            root.update()

            empty_height = (root.winfo_height() - label.winfo_height() - entry.winfo_height() - button.winfo_height()) / 4

            label.place(x=(root.winfo_width() - label.winfo_width())/2, y=empty_height)
            entry.place(x=(root.winfo_width() - entry.winfo_width()) / 2, y=2*empty_height+label.winfo_height())
            button.place(x=(root.winfo_width() - button.winfo_width()) / 2, y=3 * empty_height + label.winfo_height() + entry.winfo_height())

            root.update()

        else:
            global filename, text
            filename = filedialog.askopenfilename(
                title="Select text file",
                filetypes=[("Text files", "*.txt"), ("All files", "*.*")]
            )
            if not filename:
                filename = ""
                text = ""
```

```
                messagebox.showerror("Error", "No file selected!\nThe text is empty.")
                main(root)
            else:
                with open(filename, 'r', encoding='utf-8') as file:
                    text = file.read()
                main(root)

        try:
            [widget.destroy() for widget in root.winfo_children()]
        except Exception:
            pass
        root.title("Input the text")
        button_back = tk.Button(root, text='<-', font=main_font, command=lambda: main(root))
        button_back.place(x=10, y=10)

        combo = ttk.Combobox(root, values=["Write.", "Input from the file."], state="readonly")
        combo.set("Write.")
        button_agree= tk.Button(root, text='Agree', font=main_font, command=lambda: type(root,combo.get()))
        button_agree.pack()
        combo.pack()

        root.update()

        empty_height = (root.winfo_height() - 2 * max(combo.winfo_height(),button_agree.winfo_height())) / 3

        combo.place(x=(root.winfo_width() - combo.winfo_width())/2, y=empty_height)
        button_agree.place(x=(root.winfo_width() - button_agree.winfo_width()) / 2, y=2*empty_height+max(combo.winfo_height(),button_agree.winfo_height()))

        root.update()


    def input_key():
```

```
        def type(root, choice):
            def save(temp):
                global key
                key = temp[:-1]
                main(root)

            try:
                [widget.destroy() for widget in root.winfo_children()]
            except Exception:
                pass
            root.title("Input the key")
            button_back = tk.Button(root, text='<-', font=main_font, command=lambda: input_key())
            button_back.place(x=10, y=10)

            root.update()

            if choice == "Write.":
                label = tk.Label(root, text="Enter your key:", font=main_font)
                label.place(x=0, y=0)

                entry = tk.Text(root, width=CONST_WIDTH_TEXT, height=CONST_HEIGHT_TEXT, font=main_font)
                entry.place(x=0, y=0)

                button = tk.Button(root, text='Save', font=main_font, command=lambda: save(entry.get("1.0", tk.END)))
                button.place(x=0, y=0)

                root.update()

                empty_height = (
                                root.winfo_height() - label.winfo_height() - entry.winfo_height() - button.winfo_height()) / 4
```

```
                label.place(x=(root.winfo_width() - label.winfo_width()) / 2, y=empty_height)
                entry.place(x=(root.winfo_width() - entry.winfo_width()) / 2, y=2 * empty_height
+ label.winfo_height())
                button.place(x=(root.winfo_width() - button.winfo_width()) / 2,
                             y=3 * empty_height + label.winfo_height() + entry.winfo_height())

                root.update()

            else:
                global key
                filename = filedialog.askopenfilename(
                    title="Select text file",
                    filetypes=[("Text files", "*.txt"), ("All files", "*.*")]
                )
                if not filename:
                    key = ""
                    messagebox.showerror("Error", "No file selected!\nThe key is empty.")
                    main(root)
                else:
                    with open(filename, 'r', encoding='utf-8') as file:
                        key = file.read()
                    main(root)

        try:
            [widget.destroy() for widget in root.winfo_children()]
        except Exception:
            pass
        root.title("Input the key")
        button_back = tk.Button(root, text='<-', font=main_font, command=lambda: main(root))
        button_back.place(x=10, y=10)

        combo = ttk.Combobox(root, values=["Write.", "Input from the file."], state="readonly")
        combo.set("Write.")
        button_agree = tk.Button(root, text='Agree', font=main_font, command=lambda:
type(root, combo.get()))
```

```
        button_agree.pack()
        combo.pack()

        root.update()

        empty_height = (root.winfo_height() - 2 * max(combo.winfo_height(),
button_agree.winfo_height())) / 3

        combo.place(x=(root.winfo_width() - combo.winfo_width()) / 2, y=empty_height)
        button_agree.place(x=(root.winfo_width() - button_agree.winfo_width()) / 2,
                           y=2 * empty_height + max(combo.winfo_height(),
button_agree.winfo_height()))

        root.update()


    def aes_128():
        global text, key

        if not text or not key:
            messagebox.showerror("Error", "The text or the key are empty.")
            main(root)

        def type(root, choice):
            global text, key
            match choice:
                case "Encrypt.":
                    text = foo(text,key)
                    messagebox.showinfo("Encrypt", "The text is encrypted.")
                case "Decipher.":
                    text = foo2(text, key)
                    messagebox.showinfo("Decrypt", "The text is decrypted.")
            main(root)
```

```
        try:
            [widget.destroy() for widget in root.winfo_children()]
        except Exception:
            pass
        root.title("Input the key")
        button_back = tk.Button(root, text='<-', font=main_font, command=lambda: main(root))
        button_back.place(x=10, y=10)

        combo = ttk.Combobox(root, values=["Encrypt.", "Decipher."], state="readonly")
        combo.set("Encrypt.")
        button_agree = tk.Button(root, text='Agree', font=main_font, command=lambda:
type(root, combo.get()))
        button_agree.pack()
        combo.pack()

        root.update()

        empty_height = (root.winfo_height() - 2 * max(combo.winfo_height(),
button_agree.winfo_height())) / 3

        combo.place(x=(root.winfo_width() - combo.winfo_width()) / 2, y=empty_height)
        button_agree.place(x=(root.winfo_width() - button_agree.winfo_width()) / 2,
                           y=2 * empty_height + max(combo.winfo_height(),
button_agree.winfo_height()))

        root.update()



    def show_text_and_key():
        global text, key
```

```
        try:
            [widget.destroy() for widget in root.winfo_children()]
        except Exception:
            pass
        root.title("Show the text and the key")
        button_back = tk.Button(root, text='<-', font=main_font, command=lambda: main(root))
        button_back.place(x=10, y=10)


        label_hint = tk.Label(root, text="Your text is:", font=main_font)
        text_hint = scrolledtext.ScrolledText(root, wrap=tk.WORD, font=main_font,
                                              width=CONST_WIDTH_TEXT,
height=CONST_HEIGHT_TEXT)
        text_hint.pack(fill=tk.BOTH, expand=True)
        text_hint.insert("1.0", text)
        text_hint.config(state="disabled")
        label_key = tk.Label(root, text=f"Your key is \"{key}\".", font=main_font)

        label_hint.place(x=0, y=0)
        text_hint.place(x=0, y=0)
        label_key.place(x=0, y=0)

        root.update()


        empty_height = (root.winfo_height() - label_hint.winfo_height() -
text_hint.winfo_height() - label_key.winfo_height())/4

        label_hint.place(x=(root.winfo_width() - label_hint.winfo_width())/2, y=empty_height)
        text_hint.place(x=(root.winfo_width() - text_hint.winfo_width())/2, y=2*empty_height
+ label_hint.winfo_height())
        label_key.place(x=(root.winfo_width() - label_key.winfo_width())/2,
y=3*empty_height + label_hint.winfo_height() + text_hint.winfo_height())

        root.update()
```

```
def write_to_file():
    global filename, text

    if not filename:
        filename = filedialog.askopenfilename(
            title="Select text file",
            filetypes=[("Text files", "*.txt"), ("All files", "*.*")]
        )
        if not filename:
            filename = ""
            messagebox.showerror("Error", "No file selected!\n")
            main(root)
    else:
        filename = filename.replace('.txt','_out.txt')

    try:
        with open(filename, 'w', encoding='utf-8') as f:
            f.write(text)
        messagebox.showinfo("Success", f"Text file: \"{filename}\" - saved successfully!")
        filename = ""
    except Exception:
        messagebox.showerror("Error",f"The file: \"{filename}\" doesn't have coding utf-8.")
        main(root)



root.title("AES-128 (OVDIIENKO ANDRII)")
root.geometry("800x600")
root.resizable(False, False)
```

```
        main_font = tk.font.Font(family=font.families()[0] if "Times New Roman" not in font.families() else "Times New Roman", size=24)
        root.option_add("*Font", main_font)

        hints = [
            ["Input the text", input_text],
            ["Input the key", input_key],
            ["Use the AES-128", aes_128],
            ["Show the text and the key", show_text_and_key],
            ["Write the text in the file", write_to_file]
        ]

        buttons = []

        for hint in hints:
            buttons.append(tk.Button(root, text=hint[0], font=main_font, command=hint[1]))
            buttons[-1].pack()

        root.update()

        empty_height = (root.winfo_height() - len(buttons) * max(button.winfo_height() for button in buttons)) / (len(buttons) + 1)

        for i, button in enumerate(buttons):
            button.place(x=(root.winfo_width() - button.winfo_width())/2, y=(i+1)*empty_height + i * max(button.winfo_height() for button in buttons))
        root.update()

        root.mainloop()


    if __name__ == '__main__':
        root = tk.Tk()
        main(root)
```

4. Опис інтерфейса

Після запуска програми можна побачити головне меню.

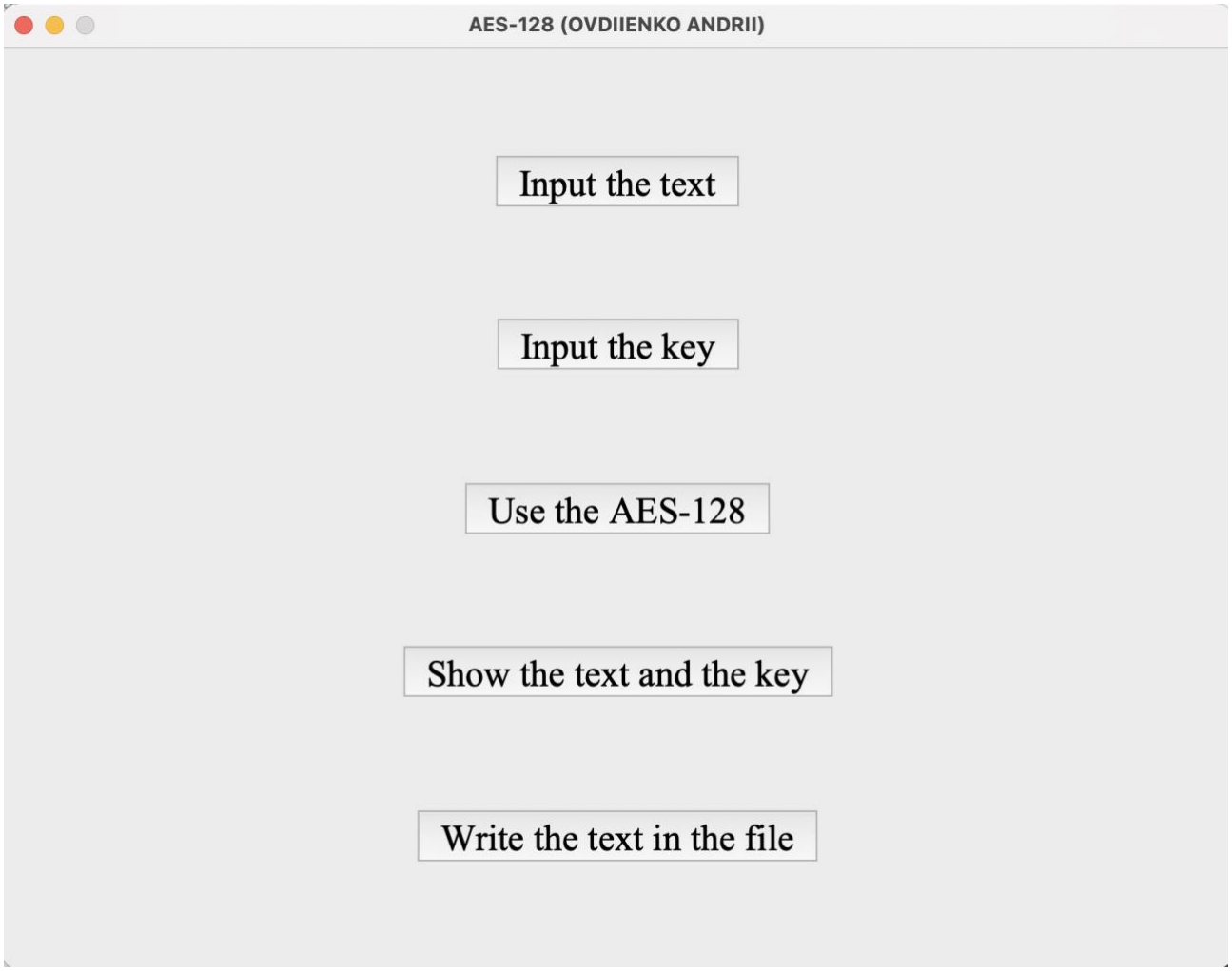

Рисунок 2 – Головне меню програми.

Кнопка “Input the text” дозволяє взяти текст із файла або написати самому.

Кнопка “Input the key” дозволяє взяти ключ із файла або написати самому.

Кнопка “Use the AES-128” дозволяє зашифрувати або розшифрувати текст.

Кнопка “Show the text and the key” – дозволяє подивитися на поточний текст та ключ.

Кнопка “Write the text in the file” – дозволяє записати текст у файл. Зауважимо, якщо текст був зчитаний з файла – то програма просто в кінці замість “.txt” підставить “_out,txt”, акщо текст був написаний самому – відкриється проводник та буде запропоновано вибрати файл для збереження.

Розмір вікна змінити не можна - це зроблено для того, щоб уникнути помилок відображення.

5. Приклад роботи програми

Відкриємо головне меню та натиснемо на кнопку для додавання текста.

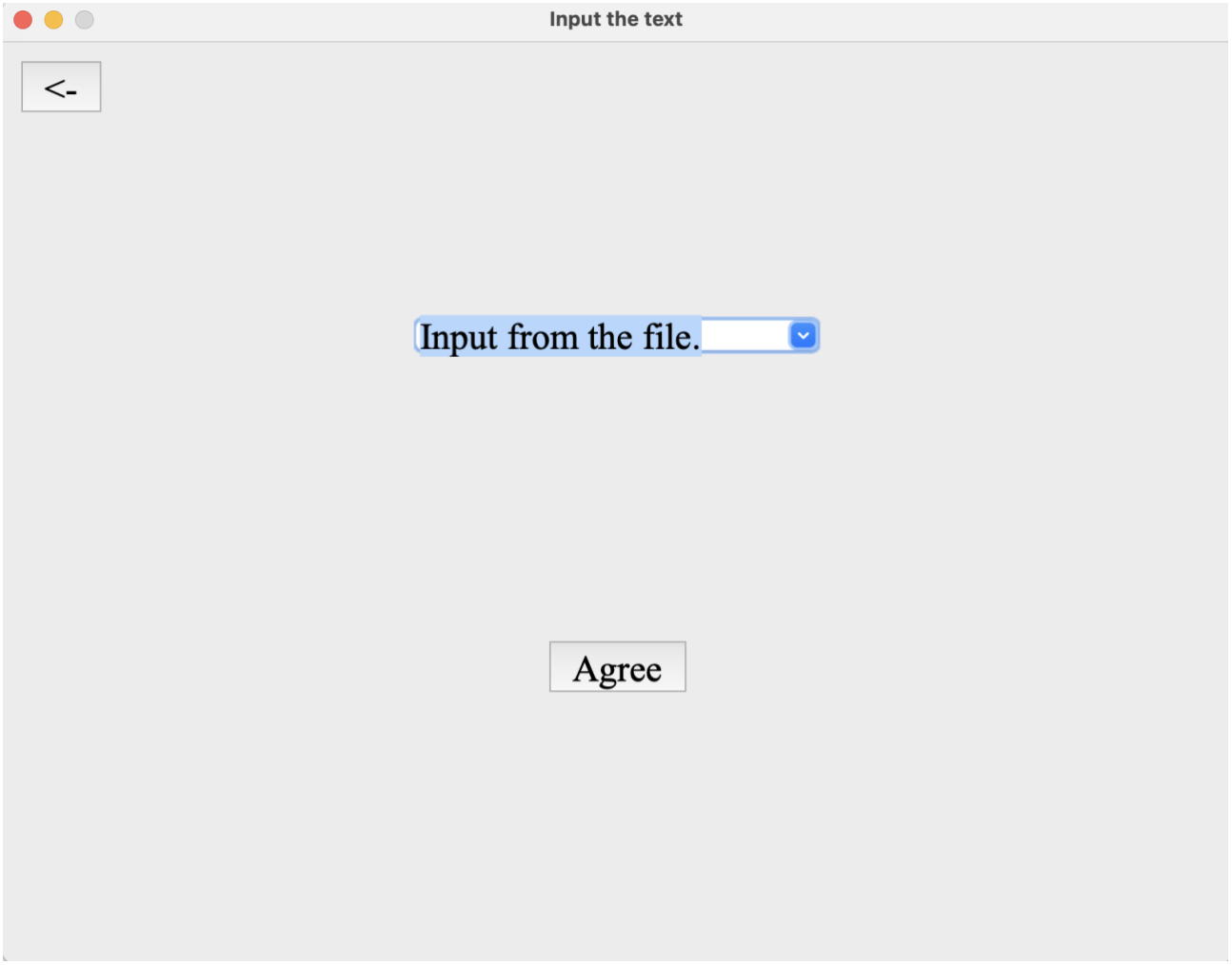

Рисунок 3 – Вибір завантаження текста з файла.

Обиремо файл.

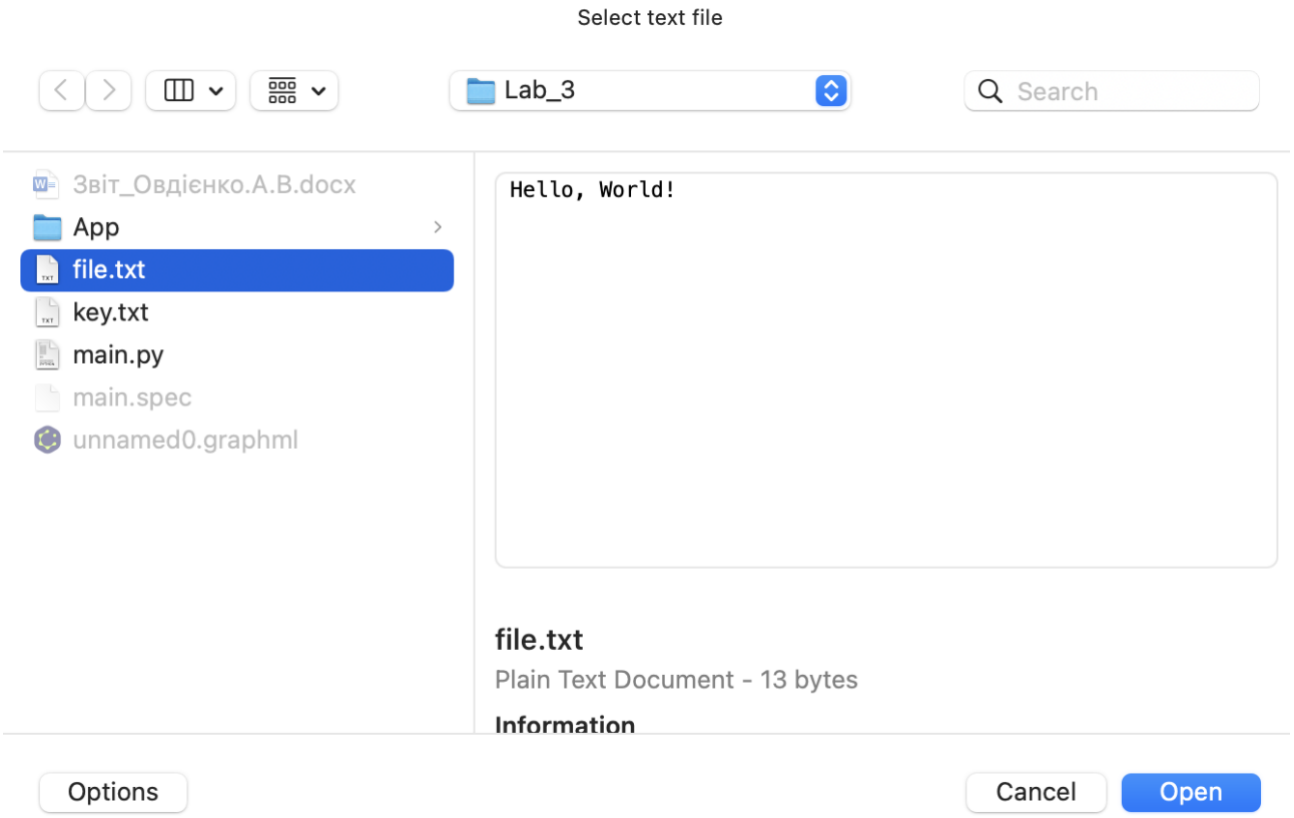

Рисунок 4 – Обираємо файл.

Натиснемо на кнопку для додавання ключа.

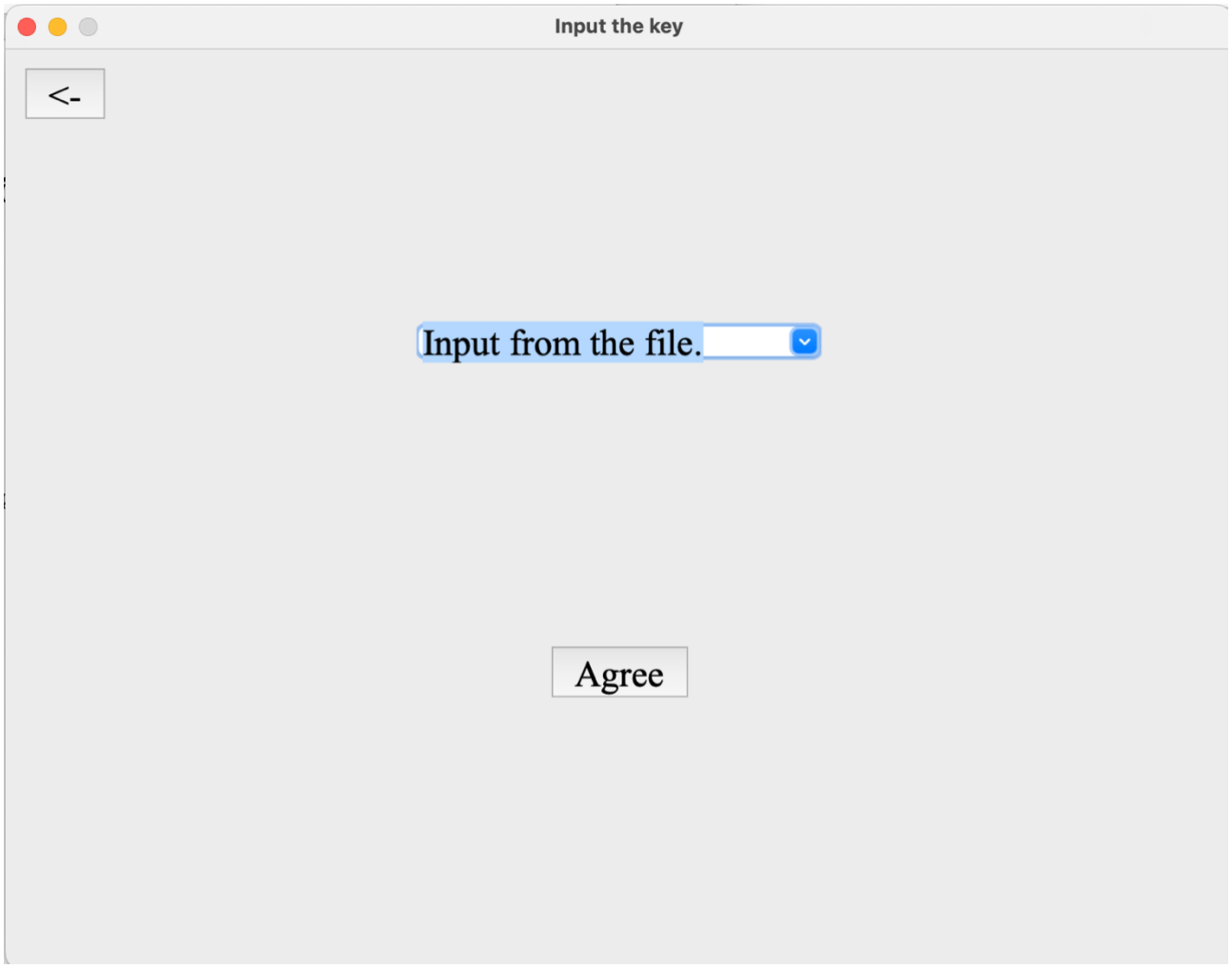

Рисунок 5 – Вибір завантаження ключа з файла.

Обиремо файл.

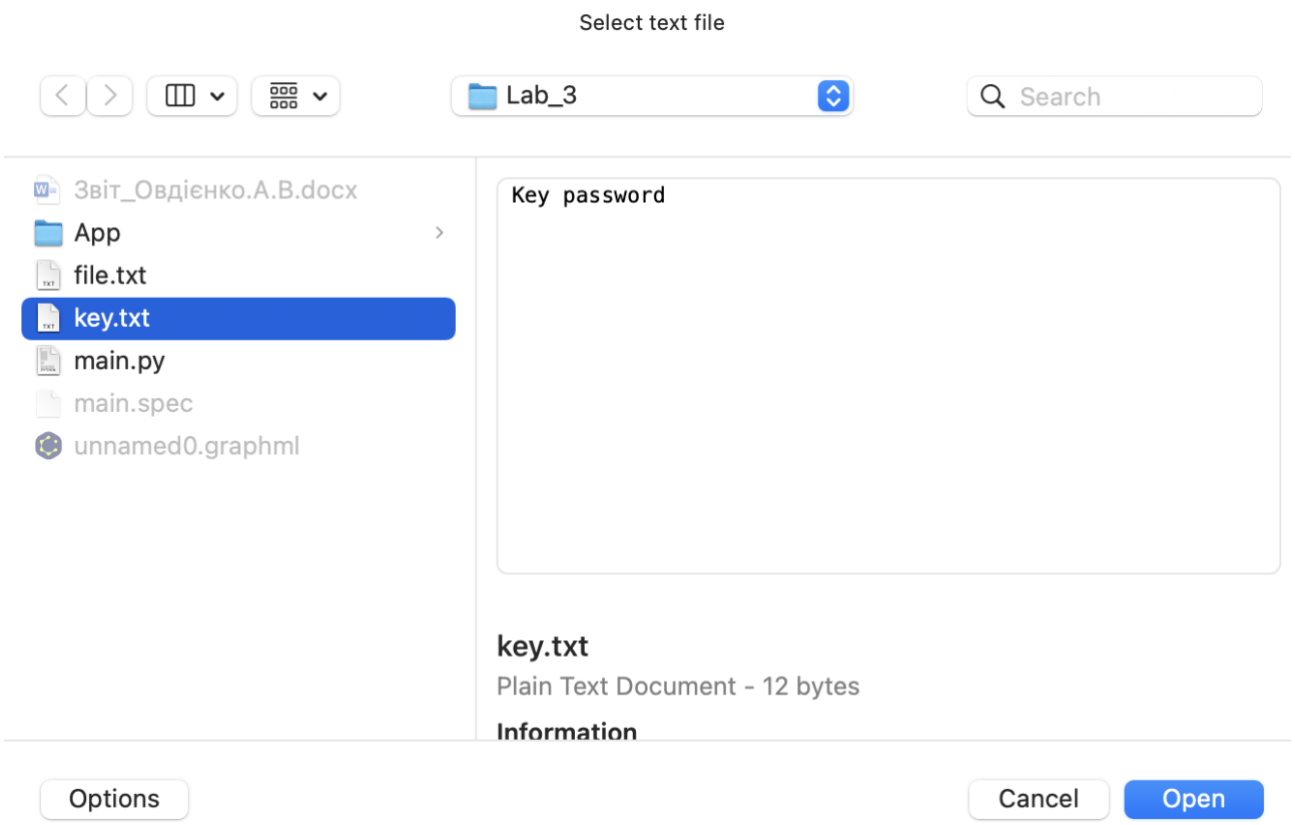

Рисунок 6 – Обираємо файл.

Натиснемо на кнопку відображеня текста та пароля.

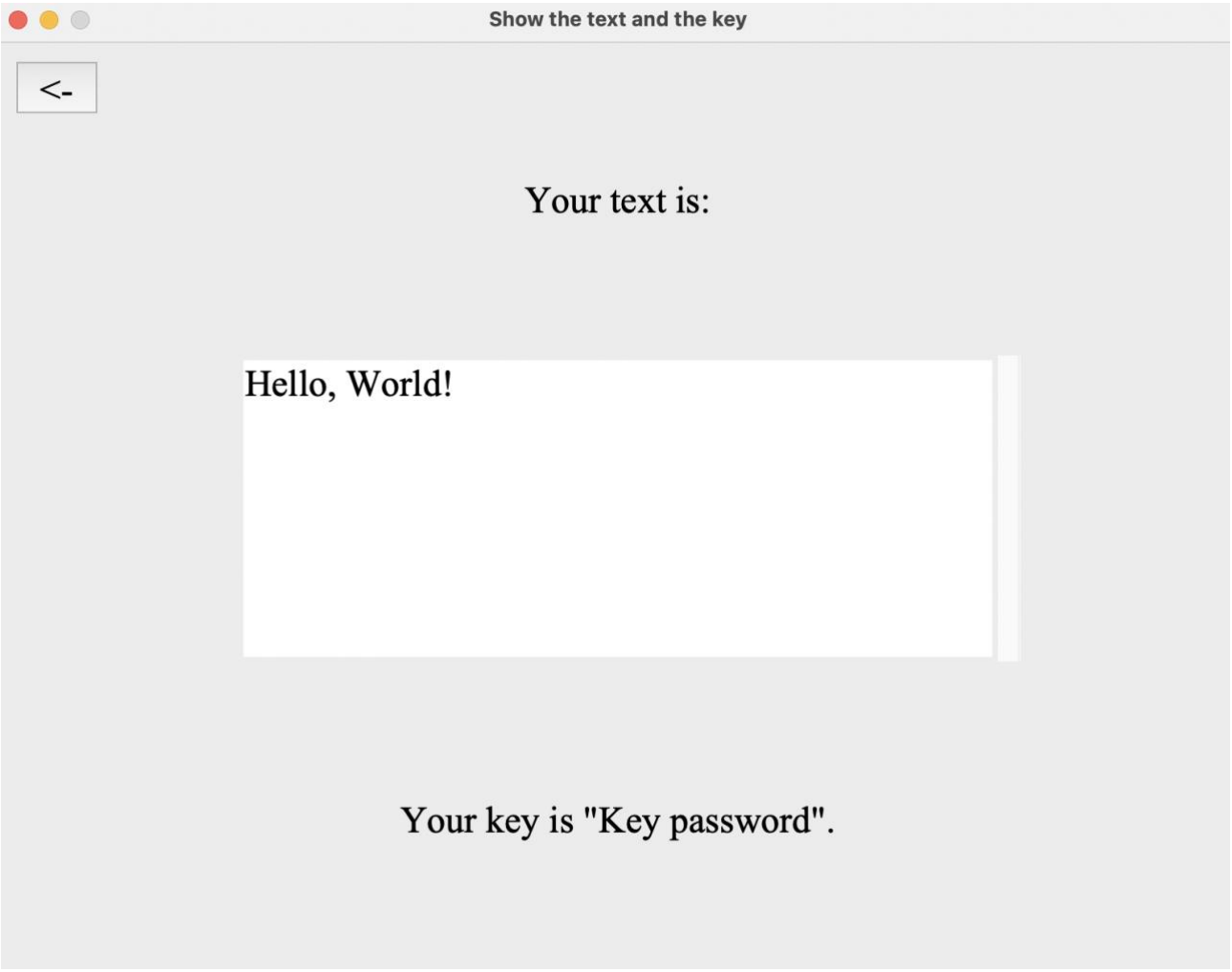

Рисунок 7 – Відображення текста та пароля.

Зашифруємо текст.

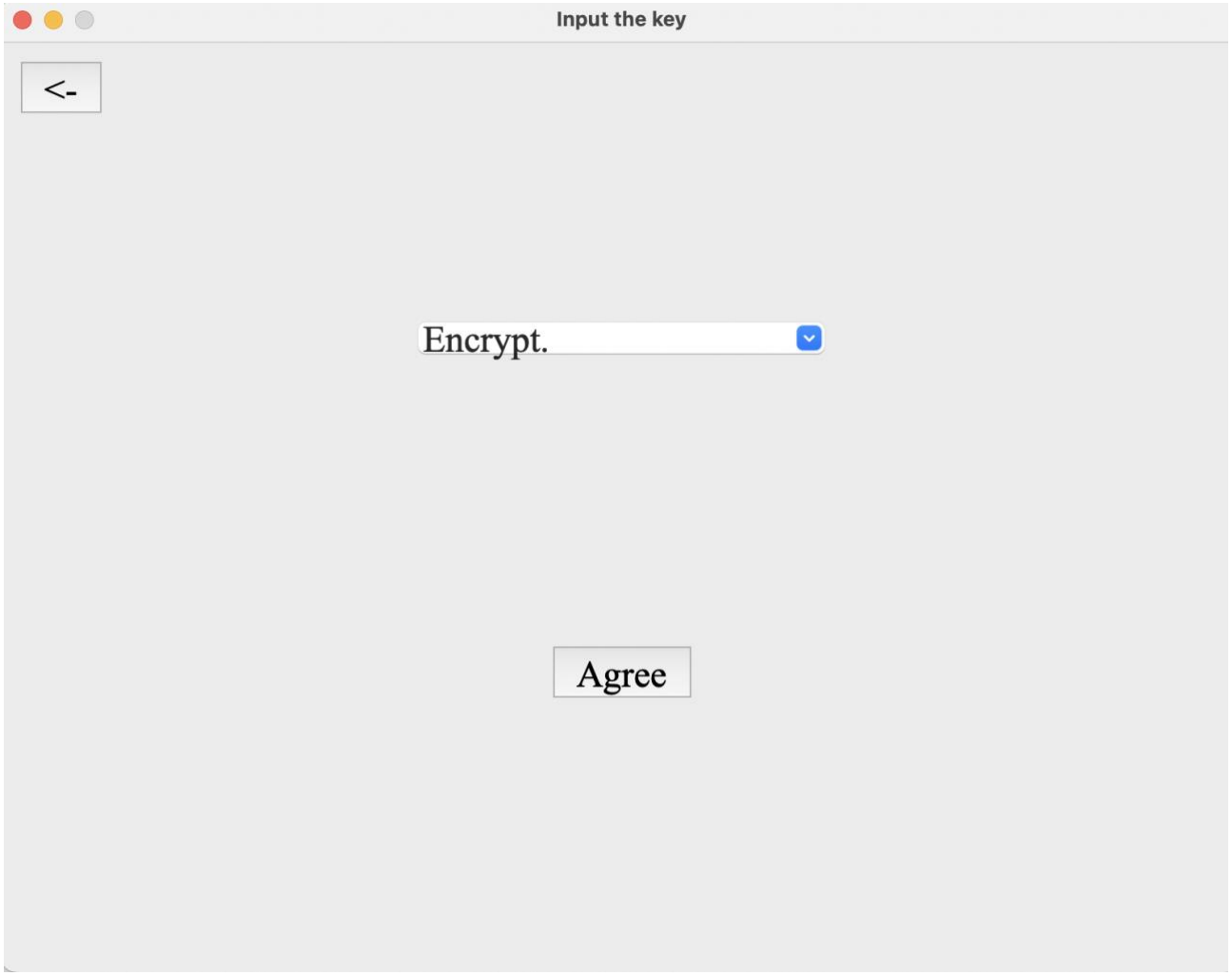

Рисунок 8 – Шифруємо текст.

Натиснемо на кнопку відображеня текста та пароля.

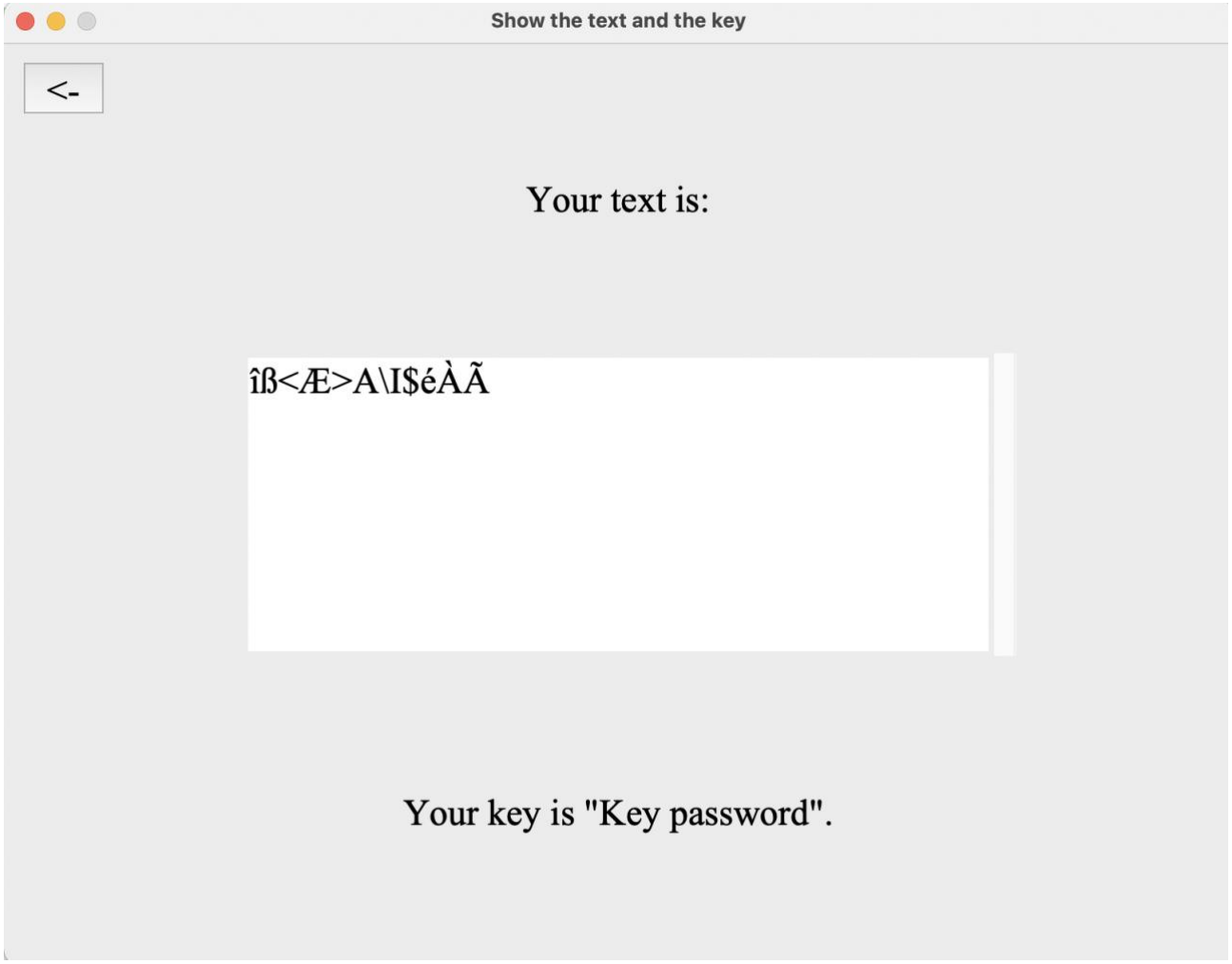

Рисунок 9 – Відображення текста та пароля.

Збережемо текст у файл. Так як брали дані з файла – то отримаємо той самий файл, але з розширенням “_out.txt”.

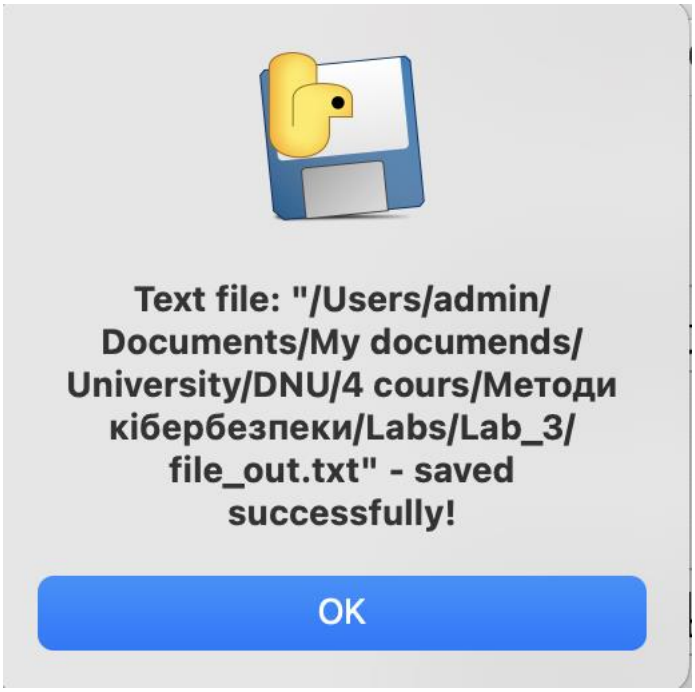

Рисунок 10 – Успішне збереження текста до файла.

А тепер зчитаємо текст з цього файла.

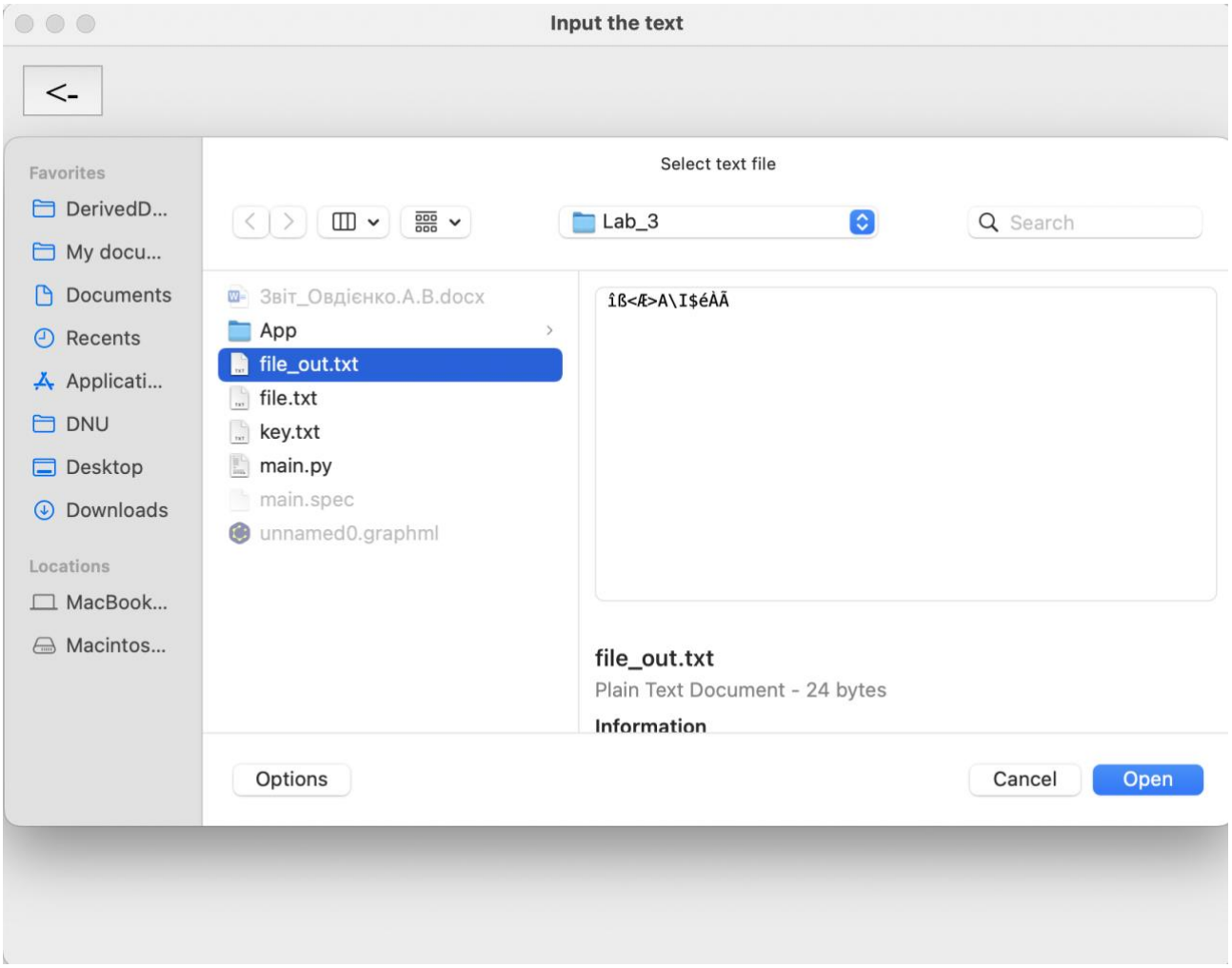

Рисунок 11 – Обираємо файл для зчитування текста.

Натиснемо на кнопку відображеня текста та пароля.

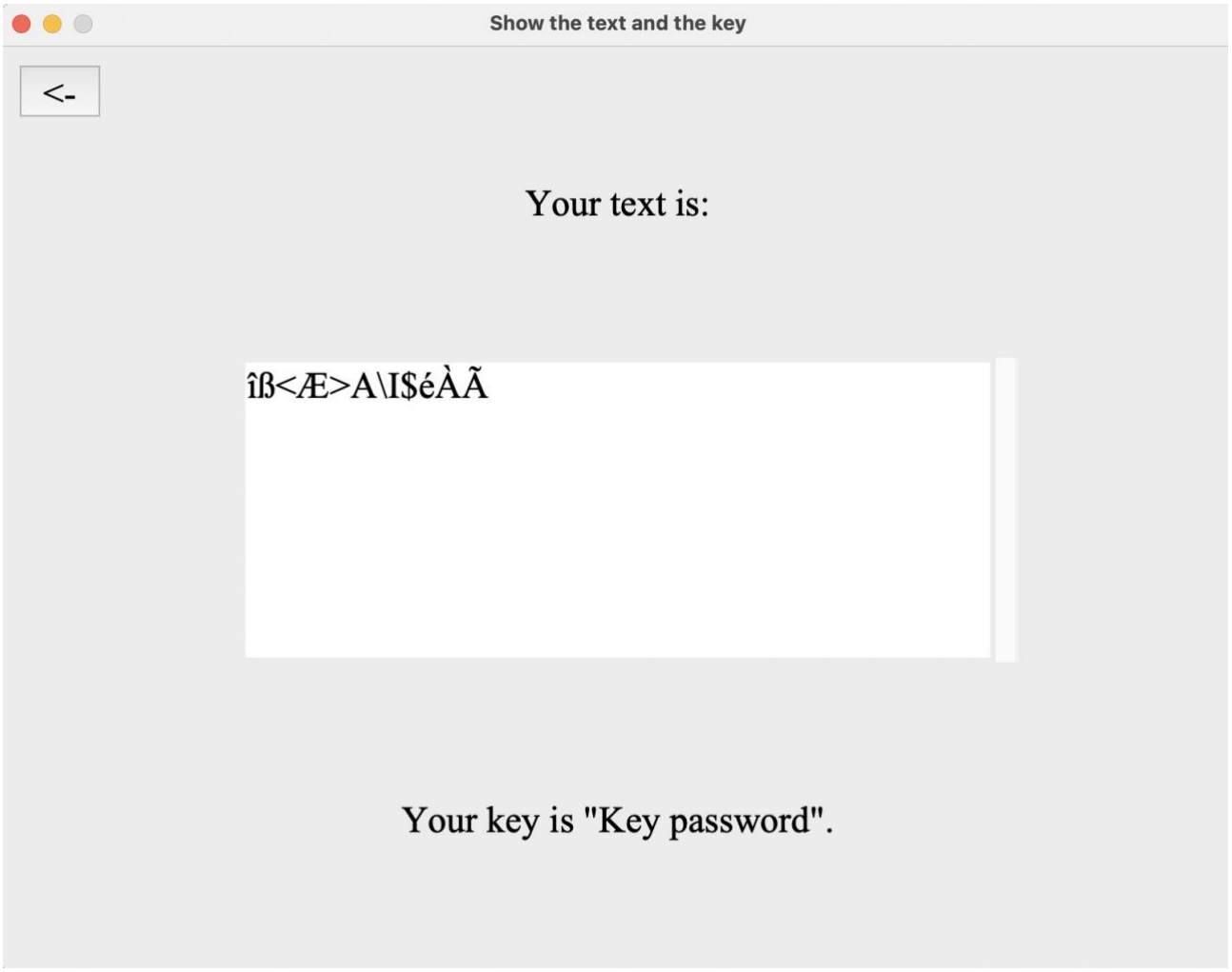

Рисунок 12 – Відображення текста та пароля.

Розшифруємо текст.

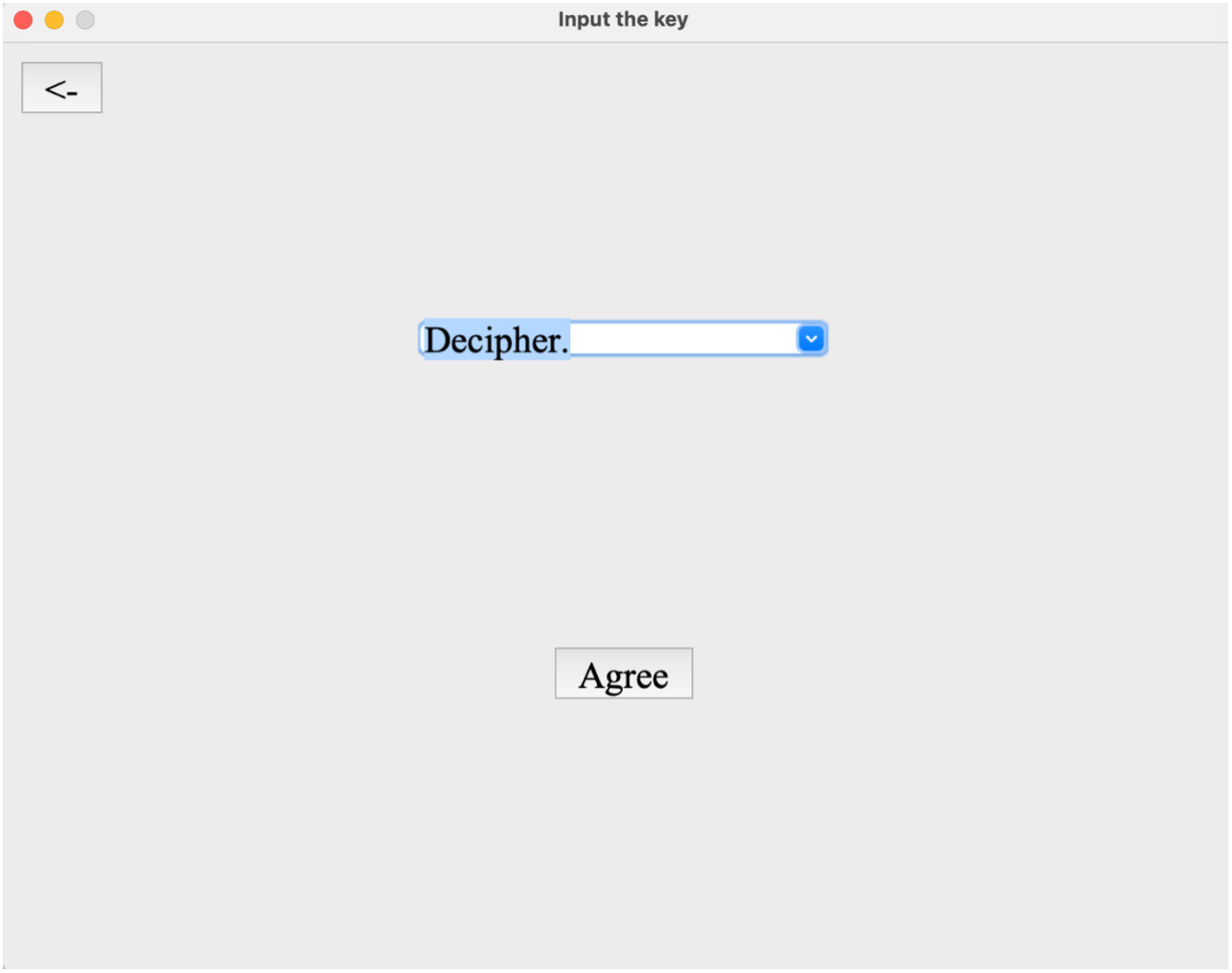

Рисунок 13 – Розшифровуємо текст.

Натиснемо на кнопку відображеня текста та пароля.

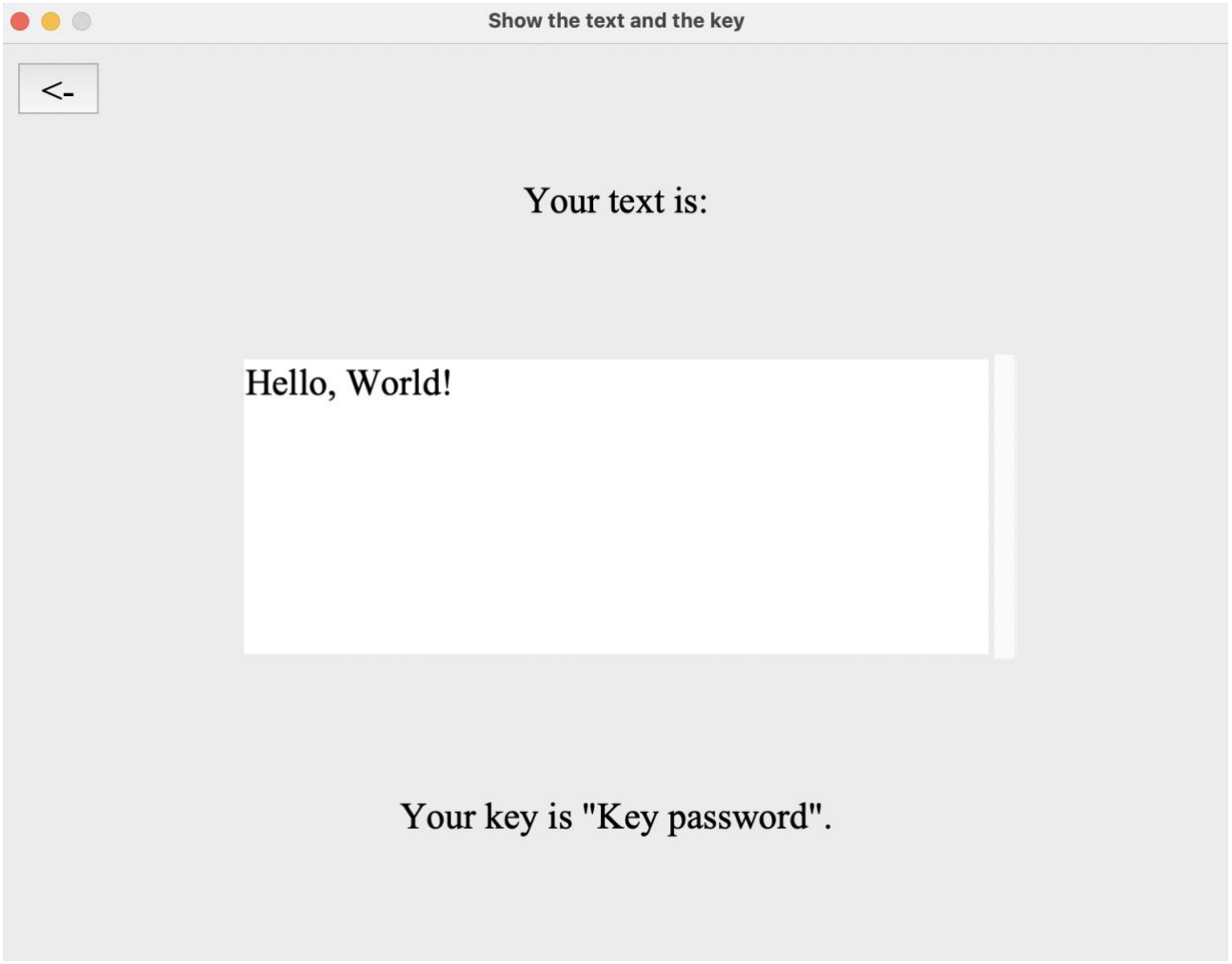

Рисунок 14 – Відображення текста та пароля.

6. Висновки

Вивчено суть симетричного алгоритма шифрування. Виявлено надзвичайну стійкість до взлома – цілих 2^{128} можливих варіантів для AES-128. Зрозуміло, що ключ має бути розміром 128 біт для данного алгоритма, а інші 10 робляться вже в ньому і шифрування відбувається з їх допомогою, що ще раз робить алгоритм стійким до підбора. Реалізовано метод шифрування AES-128 - а також дешифрування у графічному режимі та можливістю роботи з файлами. Додаток написан та скомпільован як для Mac OS так і Windows. Завантажити додатки та код можна із сайта: https://github.com/OvdiienkoAndrew/AES-128/tree/main#.