

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)
Кафедра автоматизированных систем управления (АСУ)

СИММЕТРИЧНЫЕ АЛГОРИТМЫ ШИФРОВАНИЯ

Лабораторная работа №2

по дисциплине

«Информационная безопасность»

Студент гр. 430-2

_____ А.А. Лузинсан

«_____» _____ 2023 г.

Руководитель

Ассистент каф. АСУ

_____ Я.В. Яблонский

«_____» _____ 2023 г.

Томск 2023

Оглавление

| | |
|---|----|
| 1 Цель работы..... | 3 |
| 2 Задание..... | 4 |
| 3 Описание алгоритма шифрования..... | 5 |
| 3.1 Общая информация..... | 5 |
| 3.2 Алгоритм получения раундовых ключей..... | 6 |
| 3.3 Алгоритм шифрования..... | 7 |
| 3.4 Алгоритм расшифрования..... | 9 |
| 4 Листинг программы..... | 10 |
| 5 Пример работы программы..... | 17 |
| 6 Вывод о проделанной работе..... | 18 |
| Приложение А (справочное) Определение таблиц..... | 19 |

1 Цель работы

Цель: познакомиться и научиться работать с симметричными алгоритмами шифрования.

2 Задание

Задание по варианту №9: два друга хотят обмениваться зашифрованными сообщениями, но у них нет подходящей программы. Напишите программу позволяющую шифровать и дешифровать сообщения с использованием алгоритма симметричного шифрования AES. Входные и выходные данные запишите в файл типа .txt.

3 Описание алгоритма шифрования

3.1 Общая информация

AES (англ. Advanced Encryption Standard; также Rijndael) — это симметричный алгоритм блочного шифрования, принятый в качестве стандарта шифрования правительством США по результатам конкурса AES. 26 мая 2002 года AES был объявлен стандартом шифрования.

В данной реализации используется размер блока 128 бит и ключ 128 бит.

Основные понятия, фигурируемые в данном алгоритме:

- Block — последовательность бит, оформленная в виде матрицы $4 \times N_b$, над которой осуществляются манипуляции. Каждый элемент этой матрицы представляет собой один байт. Далее форматированные байты будут представляться в шестнадцатеричном формате.
- Cipher Key — секретный криптографический ключ длиной 128 бит, который используется в методе `key_expansion` для получения раундовых ключей. Cipher Key также представляют в виде матрицы $4 \times N_k$. Все раундовые ключи будут храниться в таблице `key_schedule`.
- Ciphertext — выходные данные алгоритма шифрования.
- State — промежуточный результат шифрования, имеющий такую же форму, как и Block.
- S_box — нелинейная таблица замен (см. приложение A.1), используемая в нескольких трансформациях замены байтов при шифровании.
- Inv_S_box — нелинейная таблица замен (см. приложение A.2), используемая в нескольких трансформациях замены байтов при расшифровании.
- Rcon — массив байтов (см. приложение A.3), использующийся в

методе расширения ключа `key_expansion`.

- N_b — число столбцов, составляющих Block и State. Для AES $N_b=4$.
- N_k — число столбцов, составляющих шифроключ. Для AES в данной реализации взято значение $N_k=4$.
- N_r — число раундов. Для AES в данной реализации, относительно значения N_k , взято значение $N_r=10$.

Реализация производилась на языке Python 3.11 с использованием библиотеки `pymru`.

3.2 Алгоритм получения раундовых ключей

Первым делом формируется таблица раундовых ключей `key_schedule` в приватном методе `key_expansion()`. Первые 4 строки и 4 столбца копируются в `key_schedule`, оформляя биты сразу в виде матрицы с транспонированием, так как алгоритм подразумевает обработку матрицы не по строкам, а по столбцам. Далее производится расширение раундового ключа по следующей схеме:

1. Во временную переменную копируется предыдущий столбец (32 битное слово);
2. Если номер столбца кратен значению N_k , то временная переменная трансформируется по схеме: `sub_word(rot_word(temp)) xor Rcon[i / N_k]`, где i — номер столбца, `temp` — временная переменная. Пояснения к вспомогательным функциям будут описаны ниже.
3. Иначе, если $N_k > 6$ и $(i \bmod N_k = 4)$, то временная переменная трансформируется по схеме: `sub_word(temp)`.
4. Рассматриваемый столбец инициализируется значением: `предыдущий_столбец xor временная_переменная`.
5. Если не заполнили таблицу `key_schedule`, то переходим к рассмотрению следующего столбца и начинаем с пункта 1.

3.3 Алгоритм шифрования

Алгоритм шифрования заключается в следующем:

1. В матрицу состояния state копируется шифруемый блок на 128 бит.
2. На матрицу state накладывается первый рандомный ключ (первая часть матрицы 4x4 key_schedule) посредством метода add_round_key.
3. К матрице state применяется каскад методов: sub_bytes, shift_rows, mix_columns, add_round_key. Метод add_round_key на каждой итерации берет раундовый ключ в соответствии с номером раунда. Цикл начинается с первого (а не нулевого) раунда и заканчивается на предпоследнем раунде.
4. На последнем раунде к state также применяется каскад методов: sub_bytes, shift_rows и add_round_key. Здесь в метод add_round_key уже отправляется последний блок матрицы key_schedule.

Теперь перейдём к описанию вспомогательных методов, которые используются в методе key_expansion и в алгоритме шифрования.

Метод shift_rows работает со строками матрицы state. В этой трансформации строки матрицы циклически сдвигаются влево на r байт по горизонтали в зависимости от номера строки (начиная с 0). В итоге для одной матрицы state схема трансформации выглядит следующим образом (см. формулу 3.1):

$$\begin{array}{l}
 \text{no change} \\
 \text{left shift 1} \\
 \text{left shift 2} \\
 \text{left shift 3}
 \end{array}
 \begin{bmatrix}
 a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\
 a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\
 a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \\
 a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3}
 \end{bmatrix}
 \xrightarrow{\text{shift_rows}}
 \begin{bmatrix}
 a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\
 a_{1,1} & a_{1,2} & a_{1,3} & a_{1,0} \\
 a_{2,2} & a_{2,3} & a_{2,0} & a_{2,1} \\
 a_{3,3} & a_{3,0} & a_{3,1} & a_{3,2}
 \end{bmatrix}
 \quad (3.1)$$

Метод sub_bytes обрабатывает каждый байт состояния, независимо производя нелинейную замену байтов, используя таблицу замен. При этом таблица S_box используется на этапе шифрования, а таблица Inv_S_box на этапе расшифрования. Так как байт представляется в виде двух шестнадцатеричных чисел, то чтобы получить новое значение из таблицы замен, нужно взять элемент по строке, номер которой представлен 4 старшими разрядами байта, и столбцу, номер которого представлен 4

младшими разрядами байта. Таким образом схема трансформации выглядит следующим образом (см. формулу 3.2):

$$\begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} \end{bmatrix} \xrightarrow{\text{sub_bytes}} \begin{bmatrix} b_{0,0} & b_{0,1} & b_{0,2} & b_{0,3} \\ b_{1,0} & b_{1,1} & b_{1,2} & b_{1,3} \\ b_{2,0} & b_{2,1} & b_{2,2} & b_{2,3} \\ b_{3,0} & b_{3,1} & b_{3,2} & b_{3,3} \end{bmatrix} \quad (3.2)$$

Метод `add_round_key` получает раундовый ключ, матрицу состояния `state` и накладывает раундовый ключ на `state`. Накладывание осуществляется побитовой операцией xor каждого байта `state` с каждым байтом раундового ключа, как продемонстрировано на рисунке 3.1.

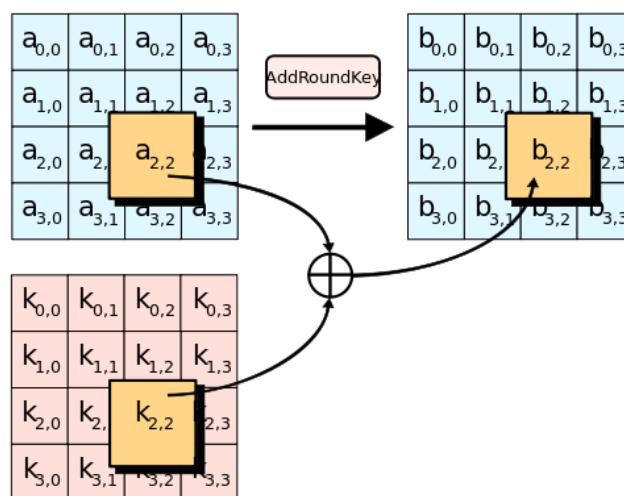


Рисунок 3.1 — Схема трансформации метода `add_round_key`

Метод `mix_columns` манипулирует, в отличие от остальных методов, над столбцами матрицы `state`. Каждая колонка представляется в виде полинома, который далее будет умножаться на другой полином. Однако, в данной реализации был взят в другой вариант представления. Чтобы получить результирующую колонку, нужно перемножить статическую матрицу на исходный столбец, как гласит формула 3.3.

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 0x02 & 0x03 & 0x01 & 0x01 \\ 0x01 & 0x02 & 0x03 & 0x01 \\ 0x01 & 0x01 & 0x02 & 0x03 \\ 0x03 & 0x01 & 0x01 & 0x02 \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix} \quad (3.3)$$

Однако в AES используется конечное поле $GF(2^8)$, для которых определены операции сложения и умножения. Операция сложения уже ранее

использовалась, так как это известная побитовая операция хог.

Для операции умножения каждый операнд, являющийся байтом, представляется в виде полинома, коэффициенты полиномов — это соответствующие биты. Непосредственно операция умножения выглядит следующим образом (см. формулу 3.4):

$$r = p * q \bmod (m) \quad (3.4)$$

где p, q — байты, представленные в виде полиномов 7 степени

m — простой полином (полином, который делится только на единицу и на самого себя), равный $m = x^8 + x^4 + x^3 + x + 1$

В перемножении полиномов p и q также должна работать определённая в поле GF операция сложения, то есть операция хог. Так как в данной реализации использовалась библиотека numru с пакетом Polynomial, то после получения результирующего полинома его коэффициенты были побитово перемножены на байт 0x01. Та же манипуляция была проделана и непосредственно с результатом r , после чего r переконвертировался к целочисленному значению байта.

3.4 Алгоритм расшифрования

Алгоритм расшифрования подобен алгоритму шифрования. Отличие состоит в том, что:

метод `shift_rows` сдвигает байты не влево, а вправо;

- метод `sub_bytes` использует таблицу замены не `S_box`, а обратную ей `Inv_S_box`;
- метод `mix_columns` использует матрицу, приведённую ниже (см. формулу 3.5):

$$\begin{bmatrix} 0x0e & 0x0b & 0x0d & 0x09 \\ 0x09 & 0x0e & 0x0b & 0x0d \\ 0x0d & 0x09 & 0x0e & 0x0b \\ 0x0b & 0x0d & 0x09 & 0x0e \end{bmatrix} \quad (3.5)$$

4 Листинг программы

Содержимое файла представлено листинге 4.1.

Листинг 4.1 — Содержимое файла по лабораторной работе

```
from typing import Union
from initialize import *
import numpy as np
import time
from numpy.polynomial import Polynomial
from collections import deque
from constant import S_BOX, R_CON, INV_S_BOX, MATRIX_ENC,
MATRIX_DEC

def get_input_data():
    if dpq.get_value('input_method') == 'File':
        file_path = dpq.get_value('file')
        file = open(file_path, 'r', encoding="utf-8")
        input_data = file.read()
    else:
        input_data = dpq.get_value('Manually')
    return input_data

class AES:
    plaintext: str
    original_length: int
    cipher_text: str
    key: str # длина ключа = 128/192/256 бит
    key_schedule: np.array
    encoding_key: str
    encoding_text: str
    blocks: np.array
    Nb: int
    Nr: int
    Nk: int
```

```

def __init__(self, key, __Nb=4, __Nr=10, __Nk=4):
    self.key = key
    self.Nb, self.Nr, self.Nk = __Nb, __Nr, __Nk

def __key_expansion():
    try:
        key_schedule = np.array(list(self.key.encode()))
        if len(key_schedule) < (self.Nk * 4): # Расширение до Nk байт
            key_schedule = np.append(key_schedule, [0x01] * ((self.Nk * 4) -
len(key_schedule)))
        elif len(key_schedule) > (self.Nk * 4):
            key_schedule = key_schedule[:self.Nk * 4]
            key_schedule = key_schedule.reshape((4, self.Nk)).transpose()
    except:
        key_schedule = np.array(np.array([[0x01] * 4]) * 4)
        dpd.set_value('key', 'Error in key')
    # Дополняем оставшиеся строки таблицы ключей
    for col in range(self.Nk, self.Nb * (self.Nr + 1)):
        temp = key_schedule[:, col - 1]
        if col % self.Nk == 0:
            temp =
AES.InternalOperations.sub_bytes(AES.InternalOperations.rot_word(temp)) \
            ^ np.array(R_CON[col // self.Nk])
        elif (self.Nk > 6) and (col % self.Nk == 4):
            temp = AES.InternalOperations.sub_bytes(temp)
            temp = key_schedule[:, col - self.Nk] ^ temp
            key_schedule = np.insert(key_schedule, col,
                np.array(temp), axis=1)
    self.key_schedule = key_schedule
    __key_expansion()

def set_plaintext(self, input_data: str):
    self.plaintext = input_data
    self.encoding_text = 'utf-16'
    self.original_length = len(self.plaintext)

```

```

def set_ciphertext(self, cipher_text: str):
    self.cipher_text = cipher_text

@staticmethod
def __split_to_blocks(text: Union[str, list], by_bytes: int):
    blocks = []
    try:
        if type(text) == str:
            text_bytes = text.encode(encoding='utf-16')
        else:
            text_bytes = text
        for i in range(0, len(text_bytes), by_bytes):
            block = np.array(list(text_bytes[i:i + by_bytes]))
            if len(block) < by_bytes:
                block = np.append(block, [0x00] * (by_bytes - len(block)))
            block = block.reshape((4, by_bytes // 4)).transpose()
            blocks.append(block)
    except Exception as err:
        print("Error in splitting a text: ", err)
    return blocks

def encrypt(self):
    encrypted_bytes = []
    num_dots = 1
    dpg.configure_item('dots', color=(0, 0, 255, 255))
    self.blocks = AES.__split_to_blocks(self.plaintext, 16)
    for block in self.blocks:
        # initializing
        state = AES.InternalOperations.add_round_key(block,
self.key_schedule[:, :self.Nb])
        for round in range(1, self.Nr):
            try:
                state = AES.InternalOperations.sub_bytes_state(state)
                state = AES.InternalOperations.shift_rows(state)
                state = AES.InternalOperations.mix_columns(state, MATRIX_ENC)
                state = AES.InternalOperations.add_round_key(state,
self.key_schedule[:,

```

```

        round * self.Nb:(round + 1) * self.Nb])
    dpg.set_value('dots', value='Encrypting: ' + ' ' * num_dots)
    num_dots += 1
except:
    print(f"Error in encrypting in {round} round of {block} block")
state = AES.InternalOperations.sub_bytes_state(state)
state = AES.InternalOperations.shift_rows(state)
state = AES.InternalOperations.add_round_key(state,
        self.key_schedule[:,
        self.Nr * self.Nb:(self.Nr + 1) * self.Nb])
state = state.transpose()
encrypted_bytes += list(state.flatten())
encrypted_data = "".join([chr(x) for x in encrypted_bytes])
return encrypted_data, encrypted_bytes

def decrypt(self, encrypted_bytes):
    decrypted_bytes = []
    num_dots = 1
    dpg.configure_item('dots', color=(0, 0, 255, 255))
    self.blocks = AES.__split_to_blocks(encrypted_bytes, 16)
    for block in self.blocks:
        # initializing
        state = AES.InternalOperations.add_round_key(block,
            self.key_schedule[:,
            self.Nr * self.Nb:self.Nb * (self.Nr + 1)])
    for round in range(self.Nr - 1, 0, -1):
        try:
            state = AES.InternalOperations.shift_rows(state, invert=True)
            state = AES.InternalOperations.sub_bytes_state(state, inverse=True)
            state = AES.InternalOperations.add_round_key(state,
                self.key_schedule[:,
                round * self.Nb:(round + 1) * self.Nb])
            state = AES.InternalOperations.mix_columns(state, MATRIX_DEC)
            dpg.set_value('dots', value='Decrypting: ' + ' ' * num_dots)
            num_dots += 1
        except:
            print(f"Error in decrypting in {round} round of {block} block")

```

```

state = AES.InternalOperations.shift_rows(state, invert=True)
state = AES.InternalOperations.sub_bytes_state(state, inverse=True)
state = AES.InternalOperations.add_round_key(state,
                                             self.key_schedule[:, :self.Nb])

state = state.transpose()
decrypted_bytes += list(state.flatten())
decrypted_data =
    bytes(decrypted_bytes).decode(encoding=self.encoding_text)
return decrypted_data[:self.original_length]

```

```
class InternalOperations:
```

```
    @staticmethod
```

```
    def sub_bytes(orig, inverse=False):
```

```
        box = INV_S_BOX if inverse else S_BOX
```

```
        return np.vectorize(lambda item: box[16 * (item // 0x10) +
                                             (item % 0x10)])(orig)
```

```
    @staticmethod
```

```
    def sub_bytes_state(state, inverse=False):
```

```
        return np.vectorize(AES.InternalOperations.sub_bytes) \
            (state, inverse=inverse)
```

```
    @staticmethod
```

```
    def rot_word(word: np.array, rotnumber=-1):
```

```
        rot = deque(word)
```

```
        rot.rotate(rotnumber)
```

```
        return np.array(rot)
```

```
    @staticmethod
```

```
    def add_round_key(state, key_schedule):
```

```
        return state ^ key_schedule
```

```
    @staticmethod
```

```
    def shift_rows(state, invert=False):
```

```
        direction = 1 if invert else -1
```

```
        new_state = state.copy()
```

```
        for index, row in enumerate(new_state):
```

```
            new_state[index] = AES.InternalOperations.rot_word \
                (row, index * direction)
```

```
        return new_state
```

```

@staticmethod
def mult_bytes(byte1, byte2):
    poly1 = Polynomial(list(map(int, list(f'{byte1:0>8b}')[::-1])))
    poly2 = Polynomial(list(map(int, list(f'{byte2:0>8b}')[::-1])))
    polynom_GF = [0x01, 0x00, 0x00, 0x00, 0x01, 0x01, 0x00, 0x01, 0x01]

    def default_list(sequence):
        sequence = map(abs, map(int, sequence))
        return list(sequence)

    mult_polys = np.bitwise_and(default_list((poly1 * poly2).coef[::-1]), 1)
    _, poly_res = np.polydiv(mult_polys, polynom_GF)
    poly_res = np.bitwise_and(default_list(poly_res), 1)
    byte_res = int(".join(map(str, poly_res)), 2)
    return byte_res

@staticmethod
def mix_columns(state, matrix):
    new_state = state.copy()
    for col in range(state.shape[1]):
        new_state[:, col] =
            [np.bitwise_xor.reduce([AES.InternalOperations.mult_bytes
                                   (state[index][col], matrix[row][index])
                                   for index
                                   in range(len(matrix[0]))])
             for row
             in range(len(matrix))]
    return new_state

def preparing(sender, app_data, user_data):
    dpg.show_item('Cipher method')
    dpg.set_value('input data', value=get_input_data())
    aes = AES(dpg.get_value('key'))
    input_data = get_input_data()
    aes.set_plaintext(input_data)
    fout = open('plaintext.txt', 'a')
    fout.writelines("\n\n\tText:\n" + input_data)

```

```

fout.close()
ciphertext, encrypted_bytes = aes.encrypt()
aes.set_ciphertext(ciphertext)
dpg.set_value('encrypted', value=ciphertext)
fout = open('ciphertext.txt', 'a')
fout.writelines("\n\tCiphertext:\n" + ciphertext)
fout.close()
# region test
decrypted_text = aes.decrypt(encrypted_bytes)
dpg.set_value('test', value=decrypted_text)
fout = open('decrypted_text.txt', 'a')
fout.writelines("\n\tDecrypted:\n" + decrypted_text)
fout.close()
if ".join(input_data) == ".join(decrypted_text):
    dpg.configure_item('dots', default_value='True', color=(0, 255, 0, 255))
else:
    dpg.configure_item('dots', default_value='False', color=(255, 0, 0, 255))
# endregion

def initialize_lr2():
    with dpg.window(label="Лабораторная работа #2", tag='lr2', show=True,
width=500, height=700, pos=(100, 100),
on_close=lambda: dpg.delete_item('lr2')):
        initialize()
        dpg.add_input_text(tag='key', label='Key', default_value='passwordfbsfv',
show=False, before='Manually')
        dpg.add_button(label="Continue: AES", callback=preparing, show=False,
tag='continue')

```


5 Пример работы программы

Программа поддерживает файловый ввод исходного текста, либо же ввод вручную, а также вывод результата в выходной файл output.txt. Результат работы для файлового ввода представлен на рисунке 5.1.

Пример работы программы на данных, введённых вручную, представлен на рисунке 5.2.

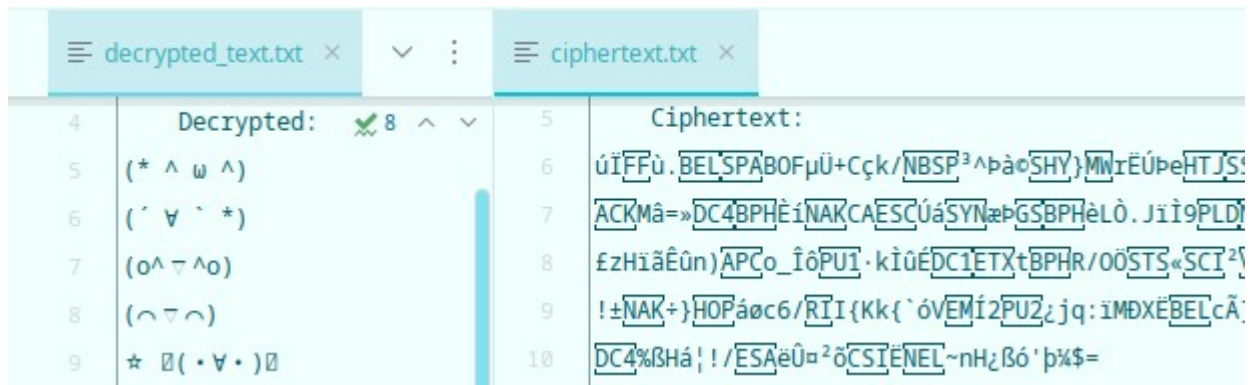


Рисунок 5.1 — Кодирование текста из файла

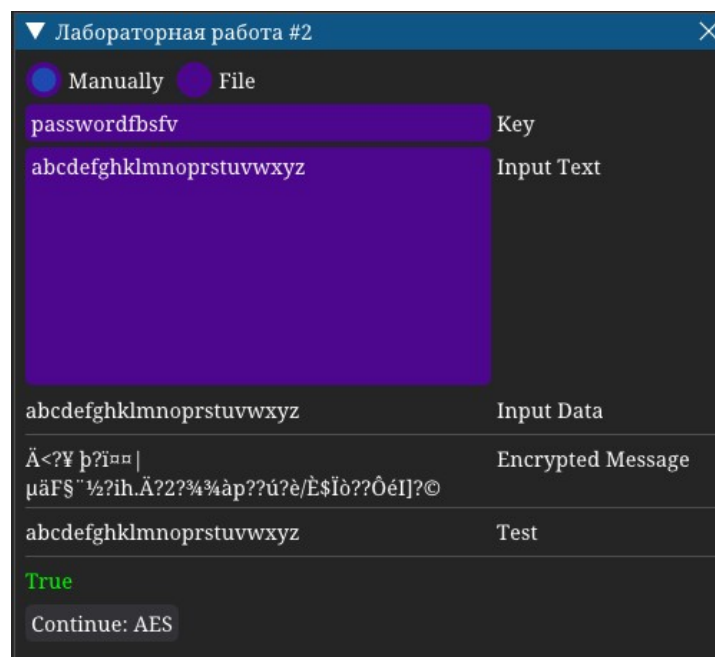


Рисунок 5.2 — Кодирование текста, введённого вручную

6 Вывод о проделанной работе

В результате выполнения лабораторной работы я познакомилась и научилась работать с симметричными алгоритмами шифрования, а также реализовала алгоритм AES.

Приложение А
(справочное)
Определение таблиц

| | | у | | | | | | | | | | | | | | | |
|---|---|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | a | b | c | d | e | f |
| х | 0 | 63 | 7c | 77 | 7b | f2 | 6b | 6f | c5 | 30 | 01 | 67 | 2b | fe | d7 | ab | 76 |
| | 1 | ca | 82 | c9 | 7d | fa | 59 | 47 | f0 | ad | d4 | a2 | af | 9c | a4 | 72 | c0 |
| | 2 | b7 | fd | 93 | 26 | 36 | 3f | f7 | cc | 34 | a5 | e5 | f1 | 71 | d8 | 31 | 15 |
| | 3 | 04 | c7 | 23 | c3 | 18 | 96 | 05 | 9a | 07 | 12 | 80 | e2 | eb | 27 | b2 | 75 |
| | 4 | 09 | 83 | 2c | 1a | 1b | 6e | 5a | a0 | 52 | 3b | d6 | b3 | 29 | e3 | 2f | 84 |
| | 5 | 53 | d1 | 00 | ed | 20 | fc | b1 | 5b | 6a | cb | be | 39 | 4a | 4c | 58 | cf |
| | 6 | d0 | ef | aa | fb | 43 | 4d | 33 | 85 | 45 | f9 | 02 | 7f | 50 | 3c | 9f | a8 |
| | 7 | 51 | a3 | 40 | 8f | 92 | 9d | 38 | f5 | bc | b6 | da | 21 | 10 | ff | f3 | d2 |
| | 8 | cd | 0c | 13 | ec | 5f | 97 | 44 | 17 | c4 | a7 | 7e | 3d | 64 | 5d | 19 | 73 |
| | 9 | 60 | 81 | 4f | dc | 22 | 2a | 90 | 88 | 46 | ee | b8 | 14 | de | 5e | 0b | db |
| | a | e0 | 32 | 3a | 0a | 49 | 06 | 24 | 5c | c2 | d3 | ac | 62 | 91 | 95 | e4 | 79 |
| | b | e7 | c8 | 37 | 6d | 8d | d5 | 4e | a9 | 6c | 56 | f4 | ea | 65 | 7a | ae | 08 |
| | c | ba | 78 | 25 | 2e | 1c | a6 | b4 | c6 | e8 | dd | 74 | 1f | 4b | bd | 8b | 8a |
| | d | 70 | 3e | b5 | 66 | 48 | 03 | f6 | 0e | 61 | 35 | 57 | b9 | 86 | c1 | 1d | 9e |
| | e | e1 | f8 | 98 | 11 | 69 | d9 | 8e | 94 | 9b | 1e | 87 | e9 | ce | 55 | 28 | df |
| | f | 8c | a1 | 89 | 0d | bf | e6 | 42 | 68 | 41 | 99 | 2d | 0f | b0 | 54 | bb | 16 |

Рисунок А.1 — Таблица S_box

| | | у | | | | | | | | | | | | | | | |
|---|---|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | a | b | c | d | e | f |
| х | 0 | 52 | 09 | 6a | d5 | 30 | 36 | a5 | 38 | bf | 40 | a3 | 9e | 81 | f3 | d7 | fb |
| | 1 | 7c | e3 | 39 | 82 | 9b | 2f | ff | 87 | 34 | 8e | 43 | 44 | c4 | de | e9 | cb |
| | 2 | 54 | 7b | 94 | 32 | a6 | c2 | 23 | 3d | ee | 4c | 95 | 0b | 42 | fa | c3 | 4e |
| | 3 | 08 | 2e | a1 | 66 | 28 | d9 | 24 | b2 | 76 | 5b | a2 | 49 | 6d | 8b | d1 | 25 |
| | 4 | 72 | f8 | f6 | 64 | 86 | 68 | 98 | 16 | d4 | a4 | 5c | cc | 5d | 65 | b6 | 92 |
| | 5 | 6c | 70 | 48 | 50 | fd | ed | b9 | da | 5e | 15 | 46 | 57 | a7 | 8d | 9d | 84 |
| | 6 | 90 | d8 | ab | 00 | 8c | bc | d3 | 0a | f7 | e4 | 58 | 05 | b8 | b3 | 45 | 06 |
| | 7 | d0 | 2c | 1e | 8f | ca | 3f | 0f | 02 | c1 | af | bd | 03 | 01 | 13 | 8a | 6b |
| | 8 | 3a | 91 | 11 | 41 | 4f | 67 | dc | ea | 97 | f2 | cf | ce | f0 | b4 | e6 | 73 |
| | 9 | 96 | ac | 74 | 22 | e7 | ad | 35 | 85 | e2 | f9 | 37 | e8 | 1c | 75 | df | 6e |
| | a | 47 | f1 | 1a | 71 | 1d | 29 | c5 | 89 | 6f | b7 | 62 | 0e | aa | 18 | be | 1b |
| | b | fc | 56 | 3e | 4b | c6 | d2 | 79 | 20 | 9a | db | c0 | fe | 78 | cd | 5a | f4 |
| | c | 1f | dd | a8 | 33 | 88 | 07 | c7 | 31 | b1 | 12 | 10 | 59 | 27 | 80 | ec | 5f |
| | d | 60 | 51 | 7f | a9 | 19 | b5 | 4a | 0d | 2d | e5 | 7a | 9f | 93 | c9 | 9c | ef |
| | e | a0 | e0 | 3b | 4d | ae | 2a | f5 | b0 | c8 | eb | bb | 3c | 83 | 53 | 99 | 61 |
| | f | 17 | 2b | 04 | 7e | ba | 77 | d6 | 26 | e1 | 69 | 14 | 63 | 55 | 21 | 0c | 7d |

Рисунок А.2 — Таблица Inv_S_box

```
[0x00, 0x00, 0x00, 0x00]  
[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]
```

Рисунок 3.3 — Таблица Rcon