МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

Федеральное государственное автономное

образовательное учреждение высшего образования

«Национальный исследовательский университет ИТМО»

**Факультет программной инженерии и компьютерной техники**

**ЛАБОРАТОРНАЯ РАБОТА №1.2**

по дисциплине

«Информационная безопасность»

Криптография. Блочное симметричное шифрование.

Вариант №5б

**Студент:**

Воробьев К.О.

            Группа P34302

**Преподаватель:**

Фамилия И.О.

A black and white logo

Description automatically generated

Санкт-Петербург, 2023

**Цель работы**

Изучение структуры и основных принципов работы современных алгоритмов блочного симметричного шифрования, приобретение навыков программной реализации блочных симметричных шифров.

**Вариант задания №5б:**

Реализовать систему симметричного блочного шифрования, позволяющую шифровать и дешифровать файл на диске с использованием заданного блочного шифра в заданном режиме шифрования. Перечень блочных шифров и режимов шифрования приведен в таблице. Алгоритм – Rijndael, режим шифрования – CBC.

**Листинг разработанной программы**

import os  
  
  
# Функция чтения данных из файла  
def read\_file(filename):  
 with open(filename, 'r', encoding='utf-8') as file:  
 text = file.read()  
 return text  
  
  
# Сохранить данные в файл  
def save\_text\_to\_file(filename, encrypted\_file):  
 with open(filename, 'w', encoding='utf-8') as output\_file:  
 output\_file.write(encrypted\_file)  
  
  
# Разделить строку на блоки по length символов  
def split\_string\_by\_length(input\_string, length):  
 return [input\_string[i:i + length] for i in range(0, len(input\_string), length)]  
  
  
# Функция генерации вектора инициализации, состоящего из 16 байт  
def initialize\_iv():  
 return os.urandom(16).hex()  
  
  
# Функция преобразования 16-байтный блок текста в матрицу типа int  
def text\_to\_int\_matrix(text):  
 hex\_list = [int(hex(ord(char))[2:], 16) for char in text]  
 matrix = [[0] \* 4 for \_ in range(4)]  
 for i in range(4):  
 for j in range(4):  
 matrix[j][i] = hex\_list[i \* 4 + j]  
 return matrix  
  
  
# Функция преобразование матрицы типа int к hex-строке  
def int\_matrix\_to\_hex\_string(int\_matrix):  
 hex\_string = ""  
 for i in range(4):  
 for j in range(4):  
 hex\_element = hex(int\_matrix[j][i])[2:].zfill(2)  
 hex\_string += hex\_element  
 return hex\_string  
  
  
# Функция преобразования матрицы типа int к тексту  
def int\_matrix\_to\_text(int\_matrix):  
 text = ""  
 for i in range(4):  
 for j in range(4):  
 char\_code = int\_matrix[j][i]  
 char = chr(char\_code)  
 text += char  
 return text  
  
  
# Функция преобразования hex строки к матрице типа int  
def hex\_string\_to\_int\_matrix(hex\_string):  
 matrix = [[0] \* 4 for \_ in range(4)]  
  
 for i in range(4):  
 for j in range(4):  
 start = 2 \* (4 \* i + j)  
 end = start + 2  
 byte\_hex = hex\_string[start:end]  
 matrix[j][i] = int(byte\_hex, 16)  
  
 return matrix  
  
  
# Выполняет XOR операцию между двумя матрицами (между блоком данных и ключом)  
def add\_round\_key(state, round\_key):  
 result\_matrix = [[0] \* 4 for \_ in range(4)]  
  
 for i in range(4):  
 for j in range(4):  
 result\_matrix[i][j] = state[i][j] ^ round\_key[i][j]  
  
 return result\_matrix  
  
  
# Заменяет каждый байт состояния на соответствующий элемент из S-Box  
def sub\_bytes(state):  
 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,  
 ]  
  
 for i in range(4):  
 for j in range(4):  
 byte = state[i][j]  
 row = (byte >> 4) & 0x0F  
 col = byte & 0x0F  
 state[i][j] = s\_box[row \* 16 + col]  
  
 return state  
  
  
# Заменяет каждый элемент состояния на инвертированный s-box  
def inv\_sub\_bytes(state):  
 inv\_s\_box = [  
 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  
 ]  
  
 for i in range(4):  
 for j in range(4):  
 try:  
 state[i][j] = inv\_s\_box[int(state[i][j], 16)]  
 except TypeError:  
 state[i][j] = inv\_s\_box[state[i][j]]  
  
 return state  
  
  
# Смещение байтов в строках блока данных. Первая строка остается без изменений, вторая сдвигается на один байт влево, третья на два байта влево, а четвертая на три байта влево.  
def shift\_rows(state):  
 for i in range(4):  
 state[i] = state[i][i:] + state[i][:i]  
 return state  
  
  
# Смещение байтов в строках блока данных. Первая строка остается без изменений, вторая сдвигается на один байт вправо, третья на два байта вправо, а четвертая на три байта вправо.  
def inv\_shift\_rows(state):  
 for i in range(1, 4):  
 state[i] = state[i][-i:] + state[i][:-i]  
 return state  
  
  
# Умножение на элементы поля Галуа  
def gf\_multiply(a, b):  
 p = 0  
 for \_ in range(8):  
 if b & 1:  
 p ^= a  
 hi\_bit\_set = a & 0x80  
 a <<= 1  
 if hi\_bit\_set:  
 a ^= 0x1b # Constant for GF(2^8)  
 b >>= 1  
  
 p &= 0xFF  
 return p  
  
  
# Преобразование над столбцами состояния  
def mix\_columns(state):  
 mix\_matrix = [  
 [0x02, 0x03, 0x01, 0x01],  
 [0x01, 0x02, 0x03, 0x01],  
 [0x01, 0x01, 0x02, 0x03],  
 [0x03, 0x01, 0x01, 0x02],  
 ]  
  
 result = [[0] \* 4 for \_ in range(4)]  
  
 for col in range(4):  
 for row in range(4):  
 result[row][col] = 0  
 for i in range(4):  
 # Умножение элемента блока данных на поля Галуа  
 result[row][col] ^= gf\_multiply(mix\_matrix[row][i], state[i][col])  
  
 return result  
  
  
# Обратные преобразования над столбцами состояния  
def inv\_mix\_columns(state):  
 inv\_mix\_columns\_matrix = [  
 [0x0e, 0x0b, 0x0d, 0x09],  
 [0x09, 0x0e, 0x0b, 0x0d],  
 [0x0d, 0x09, 0x0e, 0x0b],  
 [0x0b, 0x0d, 0x09, 0x0e]  
 ]  
  
 new\_state = [[0] \* 4 for \_ in range(4)]  
  
 for col in range(4):  
 for row in range(4):  
 result = 0  
 for i in range(4):  
 result ^= gf\_multiply(inv\_mix\_columns\_matrix[row][i], state[i][col])  
 new\_state[row][col] = int(hex(result)[2:].zfill(2), 16)  
  
 return new\_state  
  
  
# Функция раундов шифрования  
def encrypt\_alg\_round(state, key\_matrix):  
 for i in range(10):  
 state = sub\_bytes(state)  
 state = shift\_rows(state)  
 if i != 9:  
 state = mix\_columns(state)  
 state = add\_round\_key(state, key\_matrix)  
  
 return state  
  
  
# Функция шифрования  
def encrypt(text, key, iv):  
 result = ""  
 key\_matrix = text\_to\_int\_matrix(key)  
 data\_blocks = split\_string\_by\_length(text, 16)  
 for index, block in enumerate(data\_blocks):  
 if index == 0:  
 round\_key = text\_to\_int\_matrix(iv)  
 else:  
 round\_key = state  
 text\_matrix = text\_to\_int\_matrix(block)  
 state = add\_round\_key(text\_matrix, round\_key)  
 state = encrypt\_alg\_round(state, key\_matrix)  
 result += int\_matrix\_to\_hex\_string(state)  
 return result  
  
  
# Функция раундов дешифрации  
def decrypt\_alg\_round(state, key\_matrix):  
 for i in range(10):  
 state = add\_round\_key(state, key\_matrix)  
 if i != 0:  
 state = inv\_mix\_columns(state)  
 state = inv\_shift\_rows(state)  
 state = inv\_sub\_bytes(state)  
  
 return state  
  
  
# Функция дешифрации  
def decrypt(encrypted\_text, key, iv):  
 result = ""  
 key\_matrix = text\_to\_int\_matrix(key)  
 iv\_matrix = text\_to\_int\_matrix(iv)  
 data\_blocks = split\_string\_by\_length(encrypted\_text, 32)  
 data\_blocks.reverse()  
 for index, block in enumerate(data\_blocks):  
 state = hex\_string\_to\_int\_matrix(block)  
 state = decrypt\_alg\_round(state, key\_matrix)  
 if index == 1:  
 state = add\_round\_key(state, iv\_matrix)  
 else:  
 state = add\_round\_key(state, hex\_string\_to\_int\_matrix(data\_blocks[index + 1]))  
 result = int\_matrix\_to\_text(state) + result  
 return result  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 input\_text = read\_file('res/input.txt')  
 key = '0123456789abcdef'  
 iv = initialize\_iv()  
 encrypted\_text = encrypt(input\_text, key, iv)  
 save\_text\_to\_file('res/encrypted.txt', encrypted\_text)  
 decrypted\_text = decrypt(encrypted\_text, key, iv)  
 save\_text\_to\_file('res/decrypted.txt', decrypted\_text)  
 print("Текст для шифрования:", input\_text)  
 print("Зашифрованный текст:", encrypted\_text)  
 print("Расшифрованный текст:", decrypted\_text)

**Результаты выполнения**

A screenshot of a computer

Description automatically generated

*input.txt*

privetprivetprivprivetprivetpriv

*encrypted.txt*

199a1af6370e8e9c0fd213772ebde08179583d393fa42ce44a8fdf8bf2f9d4b5

*decrypted.txt*

privetprivetprivprivetprivetpriv

*ключ шифрования*

0123456789abcdef

**Вывод**

В ходе выполнения данной лабораторной работы я познакомился с методом блочного симметричного шифрования Rijndael (AES). Также, удалось реализовать программу, выполняющую шифрование данных файла на диске данным методом в режиме CBC.