Цель работы

Разработать программную реализацию стандарта шифрования FIPS-197 (AES).

Задачи работы

- провести анализ нормативного документа FIPS-197 (AES);
- на основе проведённого анализа разработать описание алгоритма;
- разработать программную реализацию алгоритма;
- проверить работоспособность алгоритма.

Ход работы

Описанная в стандарте FIPS-197 модель описывает процессы шифрования и дешифрования стандарта AES (Advanced Encryption Standard) – симметричный алгоритм блочного шифрования (размер блока 128 бит, ключ 128/192/256 бит), принятый в качестве стандарта шифрования правительством США по результатам конкурса AES.

AES является стандартом, основанным на алгоритме Rijndael. Для AES длина input (блока входных данных) и State (состояния) постоянна и равна 128 бит, а длина шифроключа K составляет 128, 192, или 256 бит. При этом исходный алгоритм Rijndael допускает длину ключа и размер блока от 128 до 256 бит с шагом в 32 бита. Для обозначения выбранных длин input, State и Cipher Key в 32-битных словах используется нотация Nb = 4 для input и State, Nk = 4, 6, 8 для Cipher Key соответственно для разных длин ключей.

В начале зашифровывания input копируется в массив State по правилу state[r,c] = input[r+4c], для $0 \leq r < 4$ и $0 \leq c < Nb$. После этого к State применяется процедура AddRoundKey(), и затем State проходит через процедуру трансформации (раунд) 10, 12, или 14 раз (в зависимости от длины ключа), при этом надо учесть, что последний раунд несколько отличается от предыдущих. В итоге, после завершения последнего раунда трансформации, State копируется в output по правилу output[r+4c]=state[r,c], для $0 \leq r < 4$ и $0 \leq c < N b$.

Процедура AddRoundKey() – трансформация при шифровании и обратном шифровании, при которой Round Key XOR’ится с State. Длина RoundKey равна размеру State (то есть если Nb = 4, то длина RoundKey равна 128 бит или 16 байт).

Процедура InvMixColumns() – трансформация при расшифровании, которая является обратной по отношению к MixColumns().

Процедура InvShiftRows() – трансформация при расшифровании, которая является обратной по отношению к ShiftRows().

Процедура InvSubBytes() – трансформация при расшифровании, которая является обратной по отношению к SubBytes().

Процедура MixColumns() – трансформация при шифровании, которая берёт все столбцы State и смешивает их данные (независимо друг от друга), чтобы получить новые столбцы.

Процедура RotWord() – функция, использующаяся в процедуре Key Expansion, которая берёт 4-байтовое слово и производит над ним циклическую перестановку.

Процедура ShiftRows() – трансформации при шифровании, которые обрабатывают State, циклически смещая последние три строки State на разные величины.

Процедура SubBytes() – трансформации при шифровании, которые обрабатывают State, используя нелинейную таблицу замещения байтов (S-box), применяя её независимо к каждому байту State.

Процедура SubWord() – функция, используемая в процедуре Key Expansion, которая берёт на входе четырёхбайтовое слово и, применяя S-box к каждому из четырёх байтов, выдаёт выходное слово.

Для программной реализации данного алгоритма шифрования использовался язык программирования Python 3.9.

Пример работы разработанного алгоритма:

Для работы алгоритма в режиме шифрования необходимы исходных блок размером 128 бит и ключ, размер которого может быть 128, 192 и 256 бит:

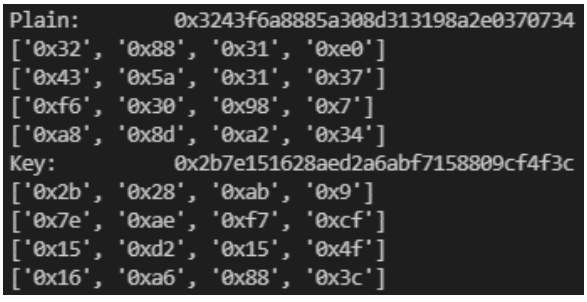

Рисунок 1 – Исходные данные

Происходит расширение ключа для генерации ключей для каждого раунда:

```
['KeyExpansion():']
['0x2b', '0x28', '0xab', '0x9']
['0x7e', '0xae', '0xf7', '0xcf']
['0x15', '0xd2', '0x15', '0x4f']
['0x16', '0xa6', '0x88', '0x3c']
['0xa0', '0x88', '0x23', '0x2a']
['0xfa', '0x54', '0xa3', '0x6c']
['0xfe', '0x2c', '0x39', '0x76']
['0x17', '0xb1', '0x39', '0x5']
['0xf2', '0x7a', '0x59', '0x73']
['0xc2', '0x96', '0x35', '0x59']
['0x95', '0xb9', '0x80', '0xf6']
['0xf2', '0x43', '0x7a', '0x7f']
['0x3d', '0x47', '0x1e', '0x6d']
['0x80', '0x16', '0x23', '0x7a']
['0x47', '0xfe', '0x7e', '0x88']
['0x7d', '0x3e', '0x44', '0x3b']
['0xef', '0xa8', '0xb6', '0xdb']
['0x44', '0x52', '0x71', '0xb']
['0xa5', '0x5b', '0x25', '0xad']
['0x41', '0x7f', '0x3b', '0x0']
['0xd4', '0x7c', '0xca', '0x11']
['0xd1', '0x83', '0xf2', '0xf9']
['0xc6', '0x9d', '0xb8', '0x15']
['0xf8', '0x87', '0xbc', '0xbc']
['0x6d', '0x11', '0xdb', '0xca']
['0x88', '0xb', '0xf9', '0x0']
['0xa3', '0x3e', '0x86', '0x93']
['0x7a', '0xfd', '0x41', '0xfd']
['0x4e', '0x5f', '0x84', '0x4e']
['0x54', '0x5f', '0xa6', '0xa6']
['0xf7', '0xc9', '0x4f', '0xdc']
['0xe', '0xf3', '0xb2', '0x4f']
['0xea', '0xb5', '0x31', '0x7f']
['0xd2', '0x8d', '0x2b', '0x8d']
['0x73', '0xba', '0xf5', '0x29']
['0x21', '0xd2', '0x60', '0x2f']
['0xac', '0x19', '0x28', '0x57']
['0x77', '0xfa', '0xd1', '0x5c']
['0x66', '0xdc', '0x29', '0x0']
['0xf3', '0x21', '0x41', '0x6e']
```

Рисунок 2 – Результат KeyExpansion()

Далее проходят процедуры AddRoundKey(), SubBytes(), ShiftRows(), MixColumns():

```
['AddRoundKey():']
['0x19', '0xa0', '0x9a', '0xe9']
['0x3d', '0xf4', '0xc6', '0xf8']
['0xe3', '0xe2', '0x8d', '0x48']
['0xbe', '0x2b', '0x2a', '0x8']
```

Рисунок 3 – Результат AddRoundKey()

```
['SubBytes():']
['0xd4', '0xe0', '0xb8', '0x1e']
['0x27', '0xbf', '0xb4', '0x41']
['0x11', '0x98', '0x5d', '0x52']
['0xae', '0xf1', '0xe5', '0x30']
```

Рисунок 4 – Результат SubBytes()

```
['ShiftRows():']
['0xd4', '0xe0', '0xb8', '0x1e']
['0xbf', '0xb4', '0x41', '0x27']
['0x5d', '0x52', '0x11', '0x98']
['0x30', '0xae', '0xf1', '0xe5']
```

Рисунок 5 – Результат ShiftRows()

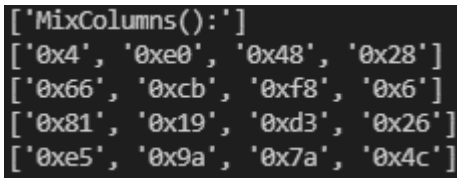

Рисунок 6 – Результат MixColumns()

В результате работы процедур на протяжении Nr раундов получаем зашифрованный блок.

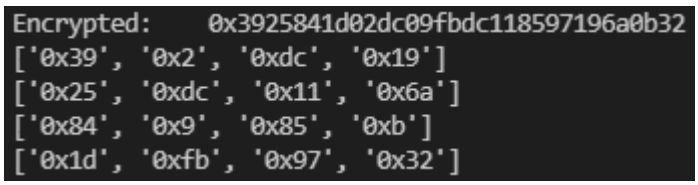

Рисунок 7 – Результат шифрования

Дешифрования проходит обратным алгоритмом с использованием процедур InvSubBytes, IntShiftRows, InvMixColumns и обратным порядком ключей.

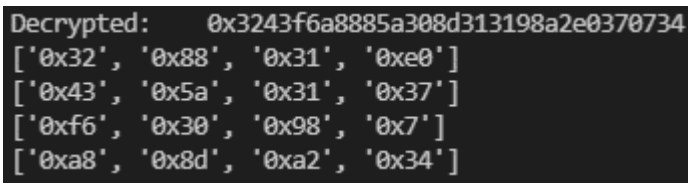

Рисунок 8 – Результат дешировния

Выводы

В данной лабораторной работе мы познакомились с алгоритмом симметричного шифрования FIPS-197 (AES), а также проанализировав работу представленного в нормативном документе алгоритма разработали программную реализацию данного алгоритма на языке Python 3.9.

Приложение А. Исходный код разработанного алгоритма.

```
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 = (0x00, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36)

# Input 128bit 3243F6A8885A308D313198A2E0370734
input_bin =
'00110010010000011111101101010100010001000010110100011000010001101001100010011000
11001100010100010111000000011011100000111001101 00'
# Key 128bit 2B7E151628AED2A6ABF7158809CF4F3C
key_bin =
'00101011011111100000101010001011000101000101011101101001010100110101010111111011
10001010110001000000010011100111101001111001 11100'
# Key 192bit 8E73B0F7DA0E6452C810F32B809079E562F8EAD2522C6B7B
# key_bin =
'10001110011100111011000011110111111011010000011100110010001010010110010000001000
01111001100101011100000001001000001111001111001010110001011111000111010101101001
00101001000101100011010110111 1011'
# Key 256bit 2B7E151628AED2A6ABF7158809CF4F3C2B7E151628AED2A6ABF7158809CF4F3C
# key_bin =
'00101011011111100000101010001011000101000101011101101001010100110101010111111011
10001010110001000000010011100111101001111001111000010101101111110000101010001011
00010100010101110110100101010011010101011111101110001010110001000000010011100111
10100111100111100'

nb = 4
nk = round(len(key_bin) / 32)
nr = round(nk + 6)
```

```
def printHex(state):
    for row in state:
        print([hex(elem) for elem in row])

def binToMatrix(plain):
    return [list(list(int(plain[row+elem:row+elem+8], 2) for row in range(0,
len(plain), 32))) for elem in range(0, 32, 8)]

def matrixToBin(matrix):
    s = ''
    for i in range(4):
        for j in range(4):
            s += format(matrix[j][i], '08b')
    return s

def addRoundKey(state, key):
    for row in range(4):
        for elem in range(4):
            state[row][elem] = state[row][elem] ^ key[row][elem]
    return state

def subBytes(state):
    for row in range(4):
        for elem in range(4):
            state[row][elem] = s_box[state[row][elem]]
    return state

def invSubBytes(state):
    for row in range(4):
        for elem in range(4):
            state[row][elem] = s_box.index(state[row][elem], 0, len(s_box))
    return state

def shiftRows(state):
    for row in range(1, 4):
        for shift in range(row):
            state[row].append(state[row].pop(0))
    return state

def invShiftRows(state):
    for row in range(1, 4):
        for shift in range(row):
            state[row].insert(0, state[row].pop())
    return state

def multiplyX2(a):
    if a < 128: a <<= 1
    else: a = (a << 1) ^ 27
    if a > 255: a %= 256
    return a

def mixMulti(a, b):
    t = a
    if b == 2:
        a = multiplyX2(a)
    elif b == 3:
        t = a
        a = multiplyX2(a)
        a ^= t
    elif b == 9:
        for i in range(3):
            a = multiplyX2(a)
        a ^= t
```

```
    elif b == 11:
            for i in range(2):
                a = multiplyX2(a)
            a ^= t
            a = multiplyX2(a)
            a ^= t
        elif b == 13:
            a = multiplyX2(a)
            a ^= t
            for i in range(2):
                a = multiplyX2(a)
            a ^= t
        elif b == 14:
            a = multiplyX2(a)
            a ^= t
            a = multiplyX2(a)
            a ^= t
            a = multiplyX2(a)
        return a

def mixColumns(s):
    t = [[0,0,0,0], [0,0,0,0], [0,0,0,0], [0,0,0,0]]
    for e in range(4):
        t[0][e] = mixMulti(s[0][e], 2) ^ mixMulti(s[1][e], 3) ^ s[2][e] ^
s[3][e]
        t[1][e] = s[0][e] ^ mixMulti(s[1][e], 2) ^ mixMulti(s[2][e], 3) ^
s[3][e]
        t[2][e] = s[0][e] ^ s[1][e] ^ mixMulti(s[2][e], 2) ^ mixMulti(s[3][e],
3)
        t[3][e] = mixMulti(s[0][e], 3) ^ s[1][e] ^ s[2][e] ^ mixMulti(s[3][e],
2)
    return t

def invMixColumns(s):
    t = [[0,0,0,0], [0,0,0,0], [0,0,0,0], [0,0,0,0]]
    for e in range(4):
        t[0][e] = mixMulti(s[0][e], 14) ^ mixMulti(s[1][e], 11) ^
mixMulti(s[2][e], 13) ^ mixMulti(s[3][e], 9)
        t[1][e] = mixMulti(s[0][e], 9) ^ mixMulti(s[1][e], 14) ^
mixMulti(s[2][e], 11) ^ mixMulti(s[3][e], 13)
        t[2][e] = mixMulti(s[0][e], 13) ^ mixMulti(s[1][e], 9) ^
mixMulti(s[2][e], 14) ^ mixMulti(s[3][e], 11)
        t[3][e] = mixMulti(s[0][e], 11) ^ mixMulti(s[1][e], 13) ^
mixMulti(s[2][e], 9) ^ mixMulti(s[3][e], 14)
    return t

def rotWord(word):
    word[0], word[1], word[2], word[3] = word[1], word[2], word[3], word[0]
    return word

def subWord(word):
    for elem in range(4):
        word[elem] = s_box[word[elem]]
    return word
```

```
def keyExpansion(key_bin):
    w = list(list(int(key_bin[row+elem:row+elem+8], 2) for elem in range(0, 32,
8)) for row in range(0, len(key_bin), 32))
    for row in range (nk, nb*(nr+1)):
        temp = w[row-1].copy()
        if row % nk == 0:
            temp = subWord(rotWord(temp))
            temp[0] = temp[0] ^ r_con[row // nk]
        elif nk > 6 and row % nk == 4:
            temp = subWord(temp)
        w.append(list(w[row-nk][elem] ^ temp[elem] for elem in range(4)))
    return list(list(list(w[key+row][elem] for row in range(4)) for elem in
range(4)) for key in range(0, nb*(nr+1), 4))

def encryption(input_bin, key_bin):
    plain_m = binToMatrix(input_bin)
    key_list = keyExpansion(key_bin)
    state = addRoundKey(plain_m, key_list[0])
    for round in range(1, nr):
        state = subBytes(state)
        state = shiftRows(state)
        state = mixColumns(state)
        state = addRoundKey(state, key_list[round])
    state = subBytes(state)
    state = shiftRows(state)
    state = addRoundKey(state, key_list[nr])
    return state

def decryption(cipher, key_bin):
    key_list = keyExpansion(key_bin)
    state = addRoundKey(cipher, key_list[nr])
    for round in range(nr-1, 0, -1):
        state = invShiftRows(state)
        state = invSubBytes(state)
        state = addRoundKey(state, key_list[round])
        state = invMixColumns(state)
    state = invShiftRows(state)
    state = invSubBytes(state)
    state = addRoundKey(state, key_list[0])
    return state

def main():
    # ENCRYPTION
    print('Plain:       ', hex(int(input_bin, 2)));
printHex(binToMatrix(input_bin))
    print('Key:         ', hex(int(key_bin, 2)));
printHex(keyExpansion(key_bin)[0])
    cipher = encryption(input_bin, key_bin)
    print('Encrypted:   ', hex(int(matrixToBin(cipher), 2))); printHex(cipher)
    # DECRYPTION
    plain_m = decryption(cipher, key_bin)
    print('Decrypted:   ', hex(int(matrixToBin(plain_m), 2))); printHex(plain_m)

if __name__ == "__main__":
    main()
```