

Цель работы:

Изучить основные принципы работы алгоритмы AES.

Задачи работы:

1. Выполнение 1 раунда алгоритма AES вручную (т.е. выполнение всех функций, входящих в раундовую функцию AES, для фиксированной входной матрицы состояний с отображением промежуточных значений шифрования) или программная реализация 1 раунда (или полной системы) AES.
2. Анализ визуализации алгоритма AES и примитивных атак на шифр, используя Cryptool 2.

Ход работы:

Часть 1.

Был программно реализован процесс шифрования для 1 раунда AES.

Модуль реализован на языке программирования Python 3.8.

Алгоритм работы программы:

1. Сложение с подключом (addRoundKey)
2. S-box (subBytes)
3. Сдвиг строк (shiftRows)
4. Перемешивание столбцов (mixColumn)

Демонстрация работы программного модуля:

1. Сгенерированное сообщение (128 бит):

```
Input:
['0x3a', '0x70', '0x56', '0xc8']
['0x92', '0xfb', '0x5e', '0xae']
['0xc4', '0x7b', '0x6', '0xdb']
['0x97', '0x0', '0xba', '0x70']
```

2. Сгенерированный ключ (128 бит):

```
Key:
['0xbe', '0x42', '0x12', '0xb1']
['0xbc', '0x3d', '0xe5', '0xb2']
['0x2f', '0xbb', '0x5a', '0x2c']
['0xa0', '0x2d', '0x77', '0x5e']
```

3. Результат выполнения функции addRoundKey():

```
addRoundKey():
['0x84', '0x32', '0x44', '0x79']
['0x2e', '0xc6', '0xbb', '0x1c']
['0xeb', '0xc0', '0x5c', '0xf7']
['0x37', '0x2d', '0xcd', '0x2e']
```

4. Результат выполнения функции subBytes():

```
subBytes():
['0x5f', '0x23', '0x1b', '0xb6']
['0x31', '0xb4', '0xea', '0x9c']
['0xe9', '0xba', '0x4a', '0x68']
['0x9a', '0xd8', '0xbd', '0x31']
```

5. Результат выполнения функции `shiftRows()`:

```
shiftRows():  
['0x5f', '0x23', '0x1b', '0xb6']  
['0xb4', '0xea', '0x9c', '0x31']  
['0x4a', '0x68', '0xe9', '0xba']  
['0x31', '0x9a', '0xd8', '0xbd']
```

6. Результат выполнения функции `mixColumns()`:

```
mixColumns():  
['0x2', '0x91', '0xb8', '0x23']  
['0xc3', '0xce', '0xc0', '0xbc']  
['0x2c', '0xac', '0x3d', '0x34']  
['0x7d', '0xc8', '0xf3', '0x2b']
```

Часть 2. Визуализация 1 раунда алгоритма AES.

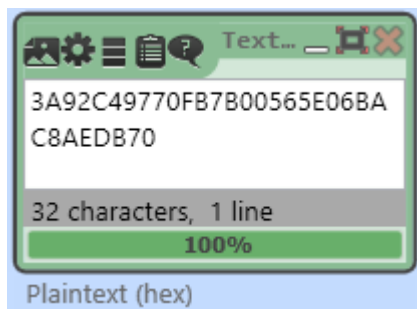
Исходное сообщение (128бит):

00111010100100101100010010010111011100001111101101111011000000000101011001011
11

0000001101011101011001000101011101101101101110000

ИЛИ

3A92C49770FB7B00565E06BAC8AEDB70

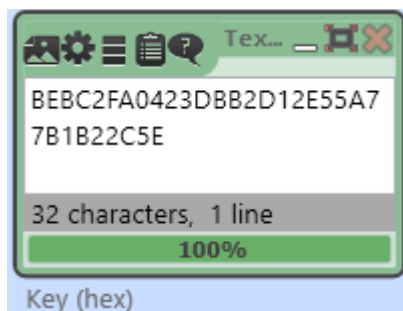


Исходный ключ (128 бит):

10111110101111000010111110100000010000100011110110111011001011010001001011100
101010110100111011110110001101100100010110001011110

ИЛИ

BEBC2FA0423DBB2D12E55A77B1B22C5E



Результат выполнения функции `addRoundKey()`:

The round key is added to the current state by XORing the bytes.

State matrix

3A	70	56	C8
92	FB	5E	AE
C4	7B	06	DB
97	00	BA	70

Round key

84	32	44	79
2E	C6	BB	1C
EB	C0	5C	F7
37	2D	CD	2E

Result matrix

BE	42	12	B1
BC	3D	E5	B2
2F	BB	5A	2C
A0	2D	77	5E

Результат выполнения функции subBytes():

Encryption **Sub Bytes** Shift Row Mix Col. Add Key

The corresponding byte in the S-box is determined and placed into the result matrix.

State matrix

84	32	44	79
2E	C6	BB	1C
EB	C0	5C	F7
37	2D	CD	2E

Result matrix

5F	23	1B	B6
31	B4	EA	9C
E9	BA	4A	68
9A	D8	BD	31

S-Box

	X0	X1	X2	X3	X4	X5	X6	X7	X8	X9	XA	XB	XC	XD	XE	XF
0X	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	A8	76
1X	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
2X	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
3X	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
4X	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
5X	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
6X	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
7X	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
8X	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
9X	60	81	4F	DC	22	2A	90	88	4E	EE	B8	14	DE	5E	08	DB
AX	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
BX	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
CX	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
DX	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
EX	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
FX	8C	A1	89	0D	8F	E6	42	68	41	99	2D	0F	B0	54	BB	16

Next Back Auto Skip Round Prev. Round Start End

To Expansion **Round 1** Round 2 Round 3 Round 4 Round 5 Round 6 Round 7 Round 8 Round 9 Round 10

Результат выполнения функции shiftRows():

Encryption Sub Bytes **Shift Row** Mix Col. Add Key

First, the second row is shifted once to the left. Then, the third row is shifted twice towards the left, and finally the forth row is shifted three times to the left. The overlapping bytes are transferred to the right to form a 4 x 4 matrix.

ShiftRow matrix

5F	23	1B	B6
B4	EA	9C	31
4A	68	E9	BA
31	9A	D8	BD

Next Back Auto Skip Round Prev. Round Start End

To Expansion **Round 1** Round 2 Round 3 Round 4 Round 5 Round 6 Round 7 Round 8 Round 9 Round 10

Результат выполнения функции mixColumns():

Encryption Sub Bytes Shift Row **Mix Col.** Add Key

Then it is multiplied with the multiplication matrix to determine the next column of the next state.

State matrix

5F	23	1B	B6
B4	EA	9C	31
4A	68	E9	BA
31	9A	D8	BD

Multiplication matrix

2	3	1	1
1	2	3	1
1	1	2	3
3	1	1	2

Transition column

02	C3	2C	7D
91	CE	AC	C8
B8	C0	3D	F3
23	BC	34	2B

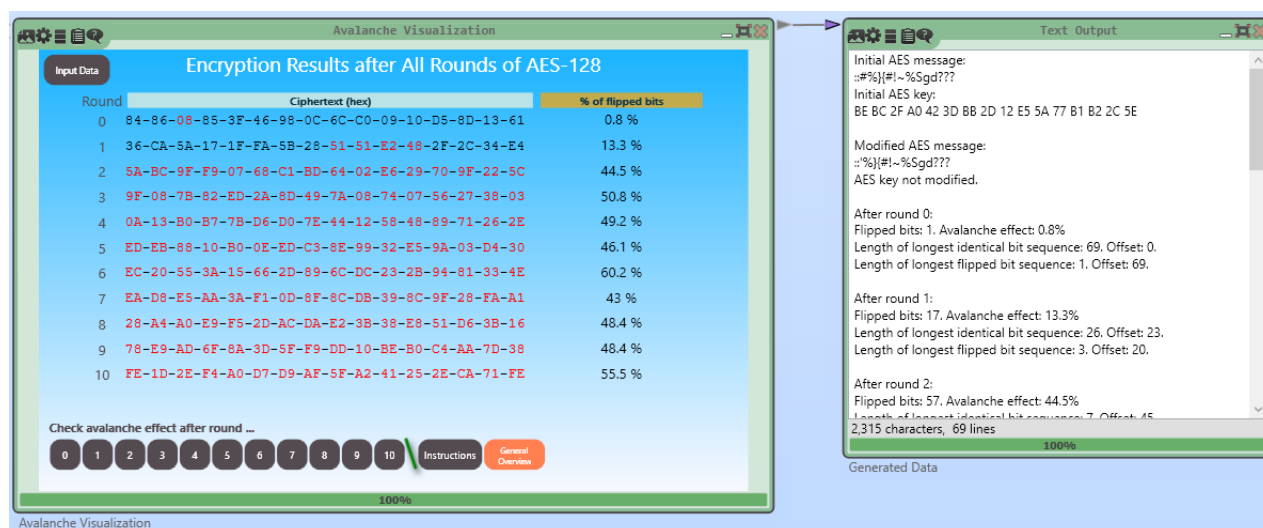
Result matrix

5F	23	1B	B6
B4	EA	9C	31
4A	68	E9	BA
31	9A	D8	BD

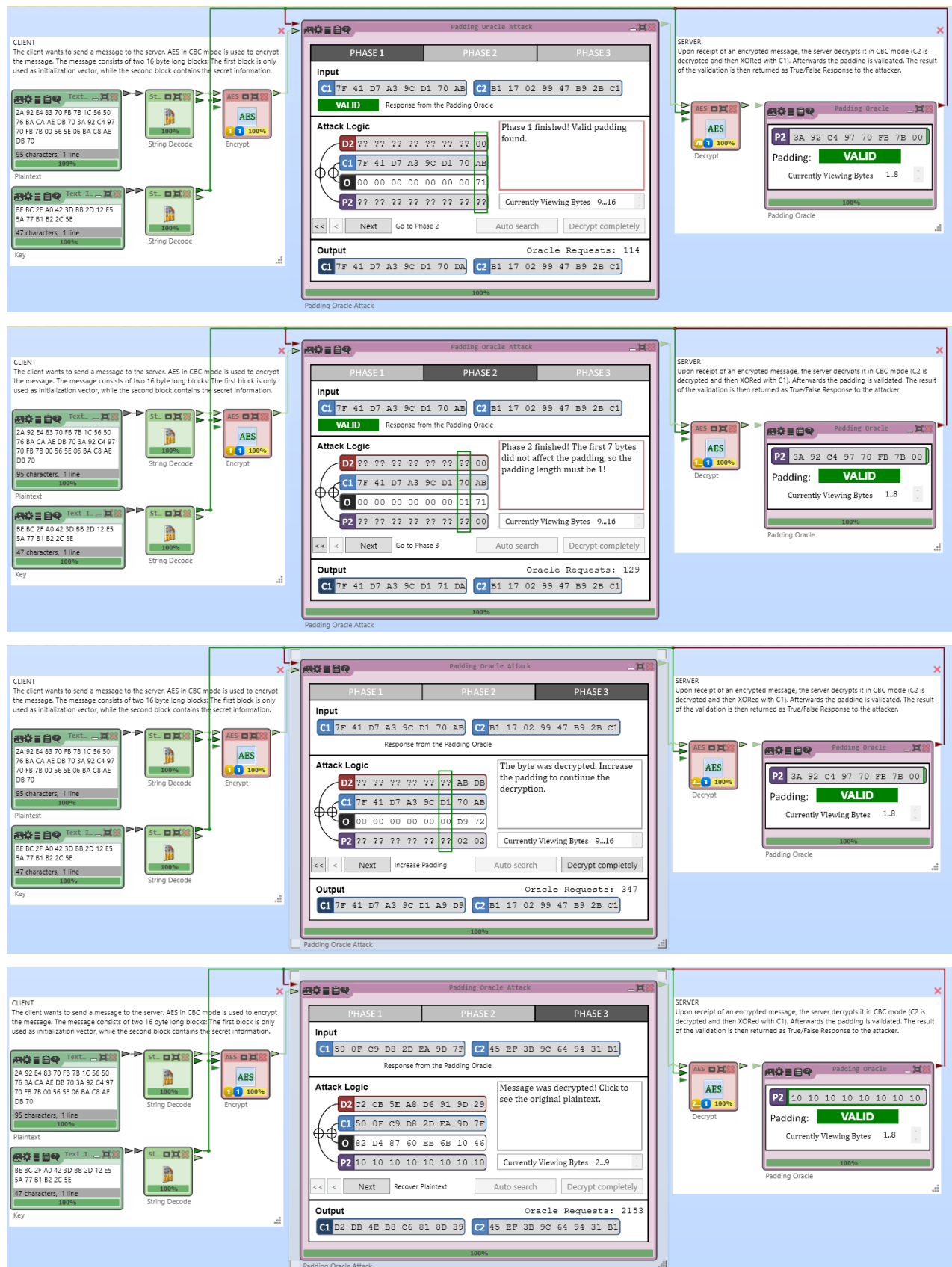
Next Back Auto Skip Round Prev. Round Start End

To Expansion **Round 1** Round 2 Round 3 Round 4 Round 5 Round 6 Round 7 Round 8 Round 9 Round 10

Изменим 1 бит в исходном сообщении и проверим, на сколько исказится шифротекст.



Часть 2. Проведение Padding Oracle атаки на DES.



Вывод:

В результате данной лабораторной работы были изучены основные принципы работы алгоритма AES, программно реализован процесс шифрования для 1 раунда AES. Был выполнен анализ визуализации алгоритма AES, продемонстрирован лавинный эффект, проведена padding oracle атака на AES.

Приложение 1. Листинг программного кода.

```
import random

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,
)

# input bin =
'0011101010010010110001001001011101110000111101101111011000000001010110010111100000011010111010110
0100010101110110101101101110000'
# key bin =
'101111101011110000101111010000001000010001111011011101100101101000100101110010101101001110111101
10001101100100010110001011110'
input_bin = ''
key_bin = ''
for i in range(128):
    input_bin += str(round(random.random()))
for i in range(128):
    key_bin += str(round(random.random()))

def printHex(state):
    for row in range(4):
        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 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 shiftRows(state):
    state[1][0], state[1][1], state[1][2], state[1][3] = state[1][1], state[1][2], state[1][3],
state[1][0]
    state[2][0], state[2][1], state[2][2], state[2][3] = state[2][2], state[2][3], state[2][0],
state[2][1]
    state[3][0], state[3][1], state[3][2], state[3][3] = state[3][3], state[3][0], state[3][1],
state[3][2]
    return state

def mixMulti(a, b):
    if b == 2:
        if a < 128: a <= 1
        else: a = (a << 1) ^ 27
        if a > 255: a %= 256
    if b == 3:
        t = a
        if a < 128: a <= 1
        else: a = (a << 1) ^ 27
        if a > 255: a %= 256
        a ^= t
    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 main():
    input_m = binToMatrix(input_bin)
    print('\nInput:')
    printHex(input_m)
    key_m = binToMatrix(key_bin)
    print('\nKey:')
    printHex(key_m)
    state = addRoundKey(input_m, key_m)
    print('\naddRoundKey():')
    printHex(state)
    state = subBytes(state)
    print('\nsubBytes():')
    printHex(state)
    state = shiftRows(state)
    print('\nshiftRows():')
    printHex(state)
    state = mixColumns(state)
    print('\nmixColumns():')
    printHex(state)

if __name__ == "__main__":
    main()

```