**Министерство науки и высшего образования Российской Федерации**

**федеральное государственное автономное образовательное учреждение высшего образования**

**«Национальный исследовательский университет ИТМО»**

**(Университет ИТМО)**

Факультет безопасности информационных технологий

Дисциплина: «Криптографические методы защиты информации»

**О Т Ч Е Т**

**Лабораторная работа № 3**

**«Основные структурные элементы алгоритма AES»**

****

|  |  |  |  |
| --- | --- | --- | --- |
| Выполнил студент группы N3351: | | Яресько С.А. | \_\_\_\_\_\_\_\_\_\_\_\_\_ |
| Проверил к.т.н., доцент ФБИТ: | Таранов С.В. | \_\_\_\_\_\_\_\_\_\_\_\_\_ |
| Дата: | \_\_\_\_\_\_\_\_\_\_ |  |

Санкт-Петербург, 2020

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.

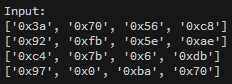
Модуль реализован на языке программирования Python 3.8.

Алгоритм работы программы:

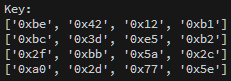
1. Cложение с подключом (addRoundKey)
2. S-box (subBytes)
3. Сдвиг строк (shiftRows)
4. Перемешивание столбцов (mixColumn)

Демонстрация работы программного модуля:

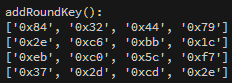
1. Сгенерированное сообщение (128 бит):



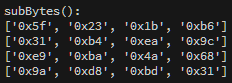
1. Сгенерированный ключ (128 бит):



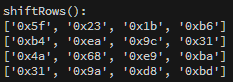
1. Результат выполнения функции addRoundKey():



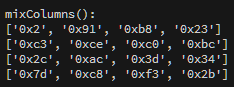
1. Результат выполнения функции subBytes():



1. Результат выполнения функции shiftRows():



1. Результат выполнения функции mixColumns():



**Часть 2. Визуализация 1 раунда алгоритма AES.**

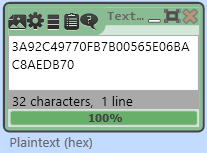
Исходное сообщение (128бит):

0011101010010010110001001001011101110000111110110111101100000000010101100101111

0000001101011101011001000101011101101101101110000

или

3A92C49770FB7B00565E06BAC8AEDB70

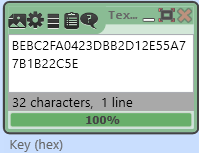


Исходный ключ (128 бит):

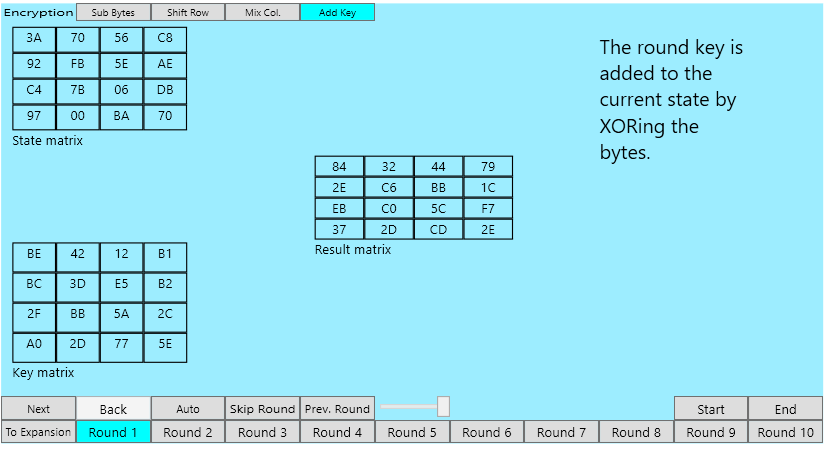
10111110101111000010111110100000010000100011110110111011001011010001001011100101010110100111011110110001101100100010110001011110

или

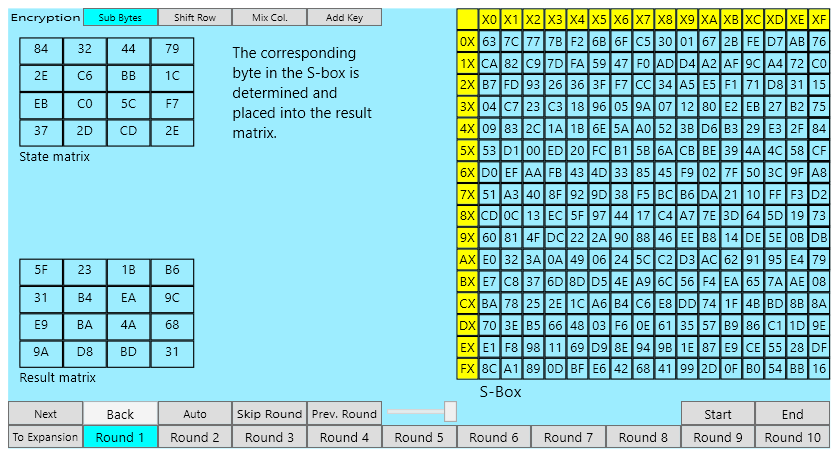
BEBC2FA0423DBB2D12E55A77B1B22C5E



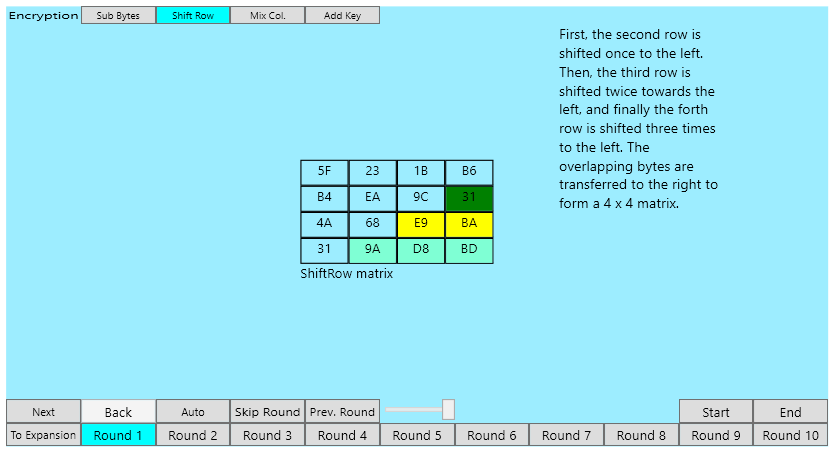
Результат выполнения функции addRoundKey():



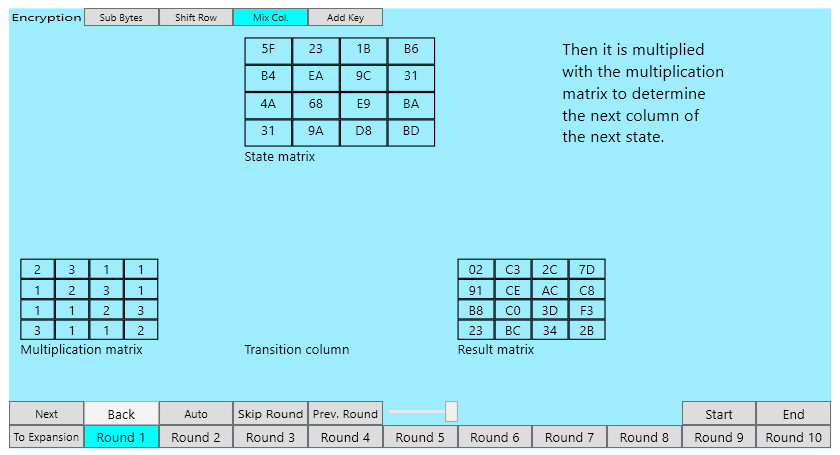
Результат выполнения функции subBytes():



Результат выполнения функции shiftRows():

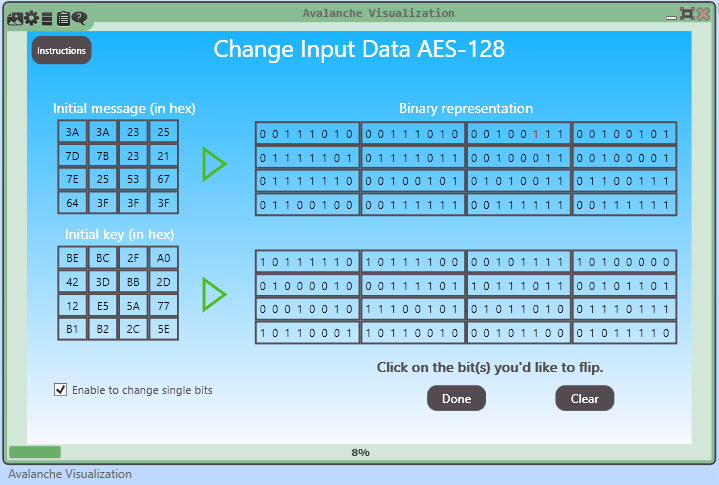


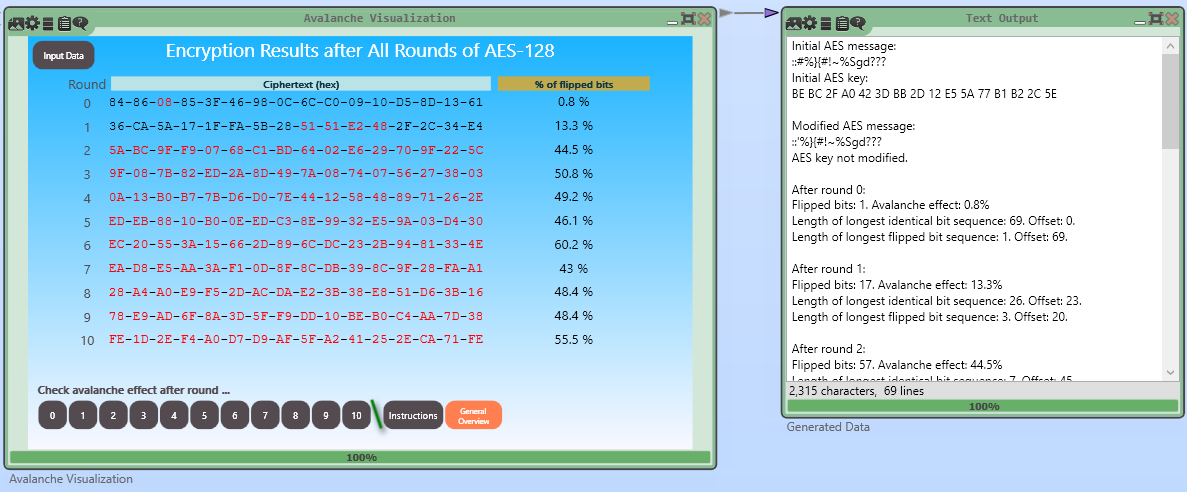
Результат выполнения функции mixColumns():



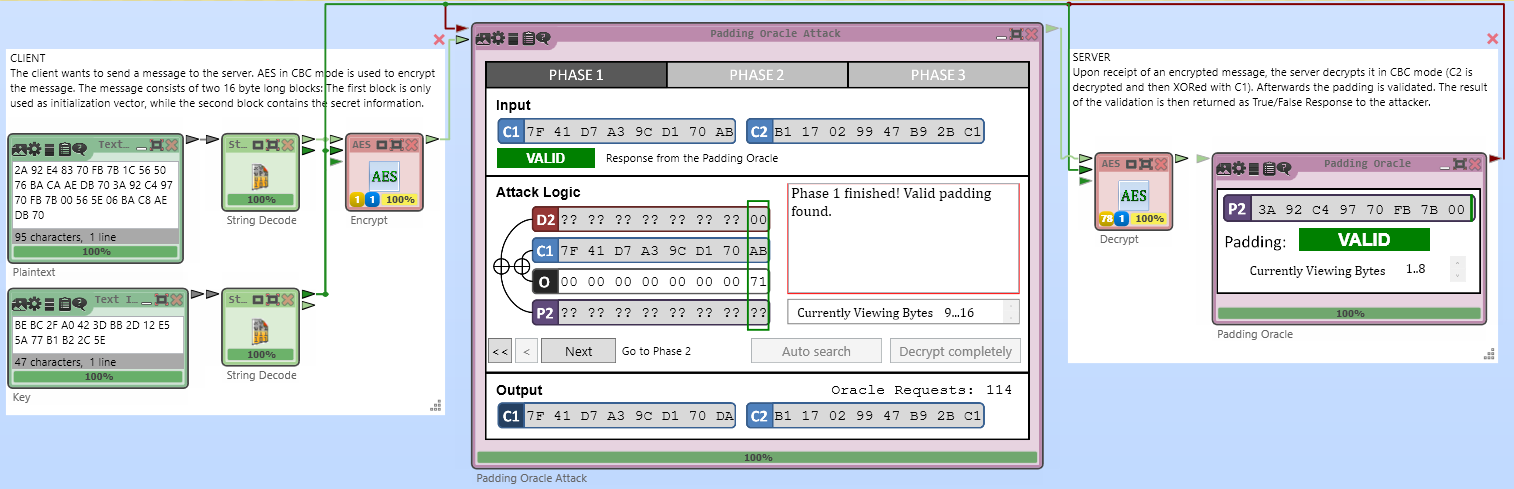
**Часть 2. Визуализация «лавинного эффекта» AES.**

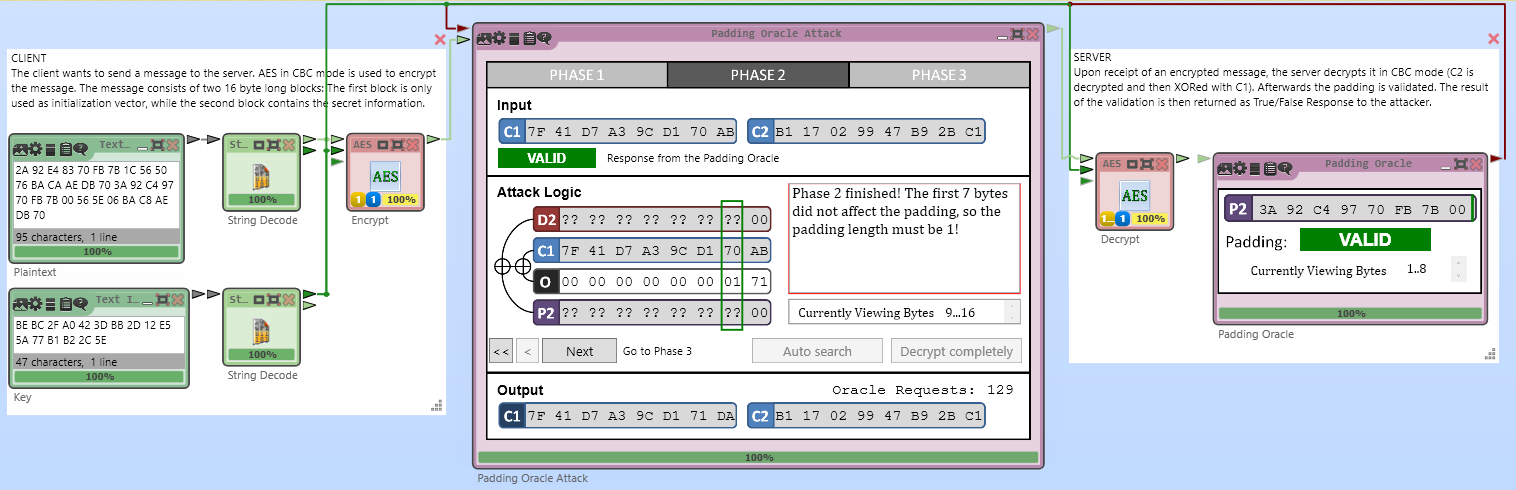
Изменим 1 бит в исходном сообщении и проверим, на сколько исказится шифротекст.

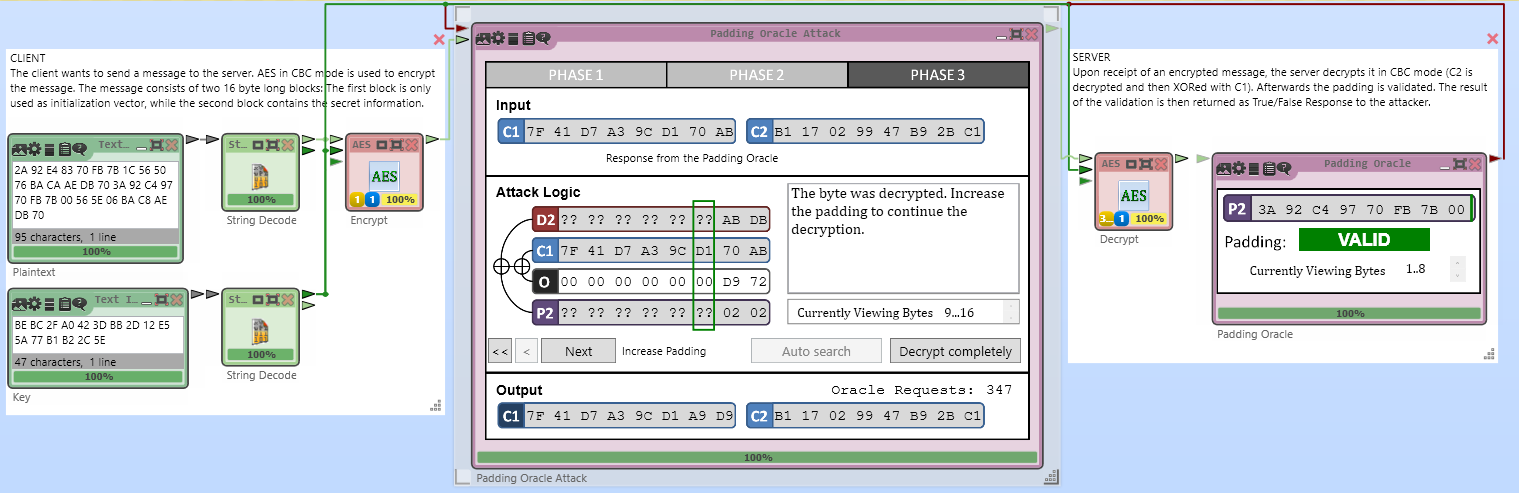


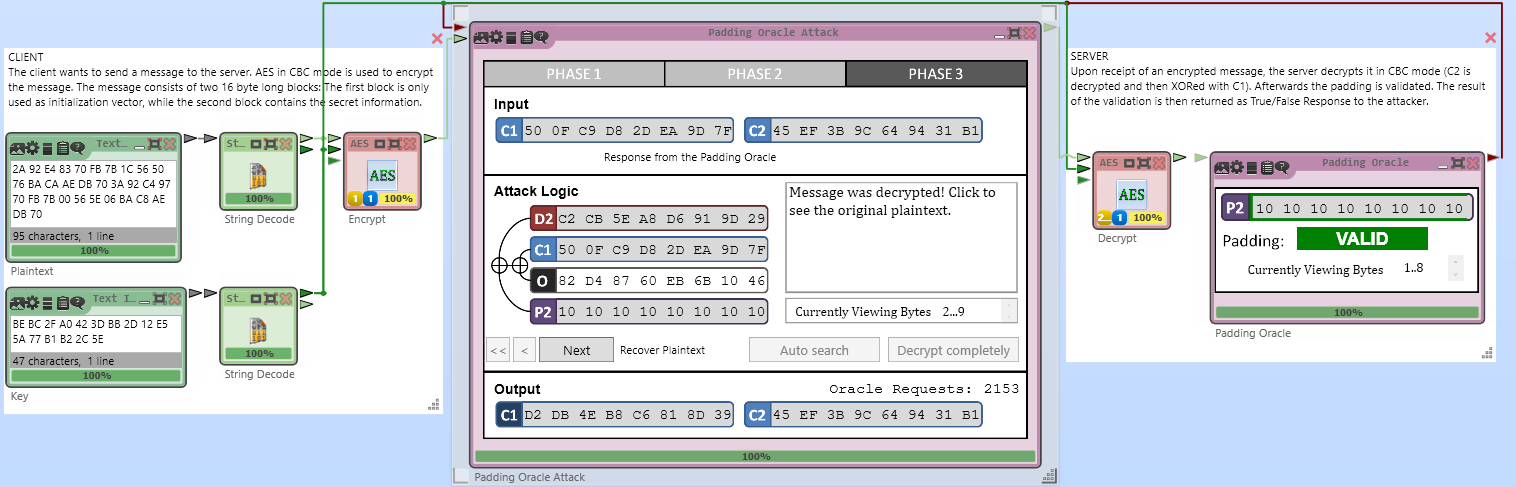


**Часть 2. Проведение Padding Oracle атаки на DES.**









Вывод:

В результате данной лабораторной работы были изучены основные принципы работы алгоритма AES, программно реализован процесса 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 = '00111010100100101100010010010111011100001111101101111011000000000101011001011110000001101011101011001000101011101101101101110000'*

*# key\_bin = '10111110101111000010111110100000010000100011110110111011001011010001001011100101010110100111011110110001101100100010110001011110'*

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** 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** 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**('**\n**Input:')

printHex(input\_m)

key\_m = binToMatrix(key\_bin)

**print**('**\n**Key:')

printHex(key\_m)

state = addRoundKey(input\_m, key\_m)

**print**('**\n**addRoundKey():')

printHex(state)

state = subBytes(state)

**print**('**\n**subBytes():')

printHex(state)

state = shiftRows(state)

**print**('**\n**shiftRows():')

printHex(state)

state = mixColumns(state)

**print**('**\n**mixColumns():')

printHex(state)

**if** \_\_name\_\_ == "\_\_main\_\_":

main()