Chelnokov Nikita, N4153c, MFMC 1.1, HW4

Task 1 (4 pts)

1) $f(x)+g(x) = x^{10}+x^9+x^8+x^5+x^4+x$

2) $f(x)-g(x) = x^{10}+x^9+x^8+x^5+x^4+x$

Similarity is caused by Z_2.

3) $f(x)g(x) = (x^{10}+x^9+x^5+x^3+1)(x^8+x^4+x^3+x+1) =$

$= x^{18}+x^{14}+x^{13}+x^{11}+x^{10}+x^{17}+x^{15}+x^{12}+x^{10}+x^9+x^{15}+x^9+x^8+x^6+x^5+x^4+x^7+x^5+x^4+x^3+$

$+x^2+x^4+x^3+x+1 = x^{18}+x^{17}+x^{14}+x^{13}+x^{12}+x^5+x+1$

4) $x^{10}+x^9+x^5+x^3+1 \;\big|\; x^8+x^4+x^3+x+1$

$x^{10}+x^6+x^5+x^3+x^2 \qquad x^2+x = q(x)$

x^6+x^2+1

$x^9+x^5+x^4+x^2+x$

$x^6+x^5+x^4+x+1 = r(x)$

5) $f(x) = x^3+1 = a(x)\cdot b(x) = (x+1)(x^2+x+1)$

$x^3+1 \;\big|\; x+1$

$x^3+x^2 \qquad x^2+x+1$

x^2+1

x^2+x

$x+1$

$x+1$

0

6) $f(x)\cdot g(x) = (x^2+x+1)(x^3+1) = x^5+x^2+x^4+x+x^3+1$

$x^5+x^4+x^3+x^2+x+1 \;\big|\; x^4+x+1$

$x^5+x^2+x \qquad x+1$

x^4+x^3+1

x^4+x+1

Result: $\boxed{x^3+x} = f(x)g(x) \bmod h(x)$

7) $\gcd(f(x), g(x))$: $x^5+x^4+1 \;\big|\; x^5+x^2+x+1$

$x^5+x^2+x+1 \qquad 1$

x^4+x^2+x

$x^5+x^2+x+1 \;\big|\; x^4+x^2+x$

$x^5+x^3+x^2 \qquad x$

$\boxed{x^3+x+1} = \gcd(f(x), g(x))$

8) $f(x) = x^7 + x^4 + x + 1 = A$; $g(x) = x^8 + x^4 + x^3 + x + 1 = N$; $A \cdot A^{-1} \equiv 1 \bmod N$; A^{-1}?

$x^8 + x^4 + x^3 + x + 1 \;|\; x^7 + x^4 + x + 1$

$-\underline{x^8 + x^5 + x^2 + x}$ $\quad x$

$x^5 + x^4 + x^3 + x^2 + 1$

I) $N = x \cdot A + (x^5 + x^4 + x^3 + x^2 + 1) \Leftrightarrow x^5 + x^4 + x^3 + x^2 + 1 = N + x \cdot A = Q$

$x^7 + x^4 + x + 1 \;|\; x^5 + x^4 + x^3 + x^2 + 1$

$-\underline{x^7 + x^6 + x^5 + x^4 + x^2}$ $\quad x^2 + x$

$x^6 + x^5 + x^2 + x + 1$

$-\underline{x^6 + x^5 + x^4 + x^3 + x}$

$x^4 + x^3 + x^2 + 1$

II) $A = (x^5 + x^4 + x^3 + x^2 + 1)(x^2 + x) + (x^4 + x^3 + x^2 + 1) \Leftrightarrow x^4 + x^3 + x^2 + 1 = A + (x^2 + x)Q$

$x^5 + x^4 + x^3 + x^2 + 1 \;|\; x^4 + x^3 + x^2 + 1$

$-\underline{x^5 + x^4 + x^3 + x}$ $\quad x$

$x^2 + x + 1$

III) $x^5 + x^4 + x^3 + x^2 + 1 = (x^4 + x^3 + x^2 + 1) \cdot x + (x^2 + x + 1) \Leftrightarrow x^2 + x + 1 = Q + x(A + (x^2 + x)Q)$

$x^4 + x^3 + x^2 + 1 \;|\; x^2 + x + 1$

$-\underline{x^4 + x^3 + x^2}$ $\quad x^2$

1

IV) $x^4 + x^3 + x^2 + 1 = (x^2 + x + 1) \cdot x^2 + 1 \Leftrightarrow 1 = A + (x^2 + x)Q + x^2(Q + x(A + (x^2 + x)Q))$

$1 = A + (x^2 + x)(N + xA) + x^2(N + xA + x(A + (x^2 + x)(N + xA))) =$

$= A + x^2N + xN + x^3A + x^2A + x^2(N + xA + x(A + x^2N + xN + x^3A + x^2A)) =$

$= A + x^2N + xN + x^3A + x^2A + x^2(N + xA + xA + x^3N + x^2N + x^4A + x^3A) =$

$= \underline{A} + x^2N + xN + \underline{x^3A} + \underline{x^2A} + x^2N + \cancel{x^3A} + \cancel{x^3A} + x^5N + x^4N + \underline{x^6A} + \underline{x^5A} \Rightarrow$

$\Rightarrow A^{-1} = \underline{x^6 + x^5 + x^3 + x^2 + 1}$

Let's check it: $A \cdot A^{-1} = (x^7 + x^4 + x + 1)(x^6 + x^5 + x^3 + x^2 + 1) =$

$= x^{13} + x^{12} + \cancel{x^{10}} + \cancel{x^8} + \cancel{x^7} + \cancel{x^{10}} + \cancel{x^8} + \cancel{x^7} + \cancel{x^6} + \cancel{x^4} + x^7 + \cancel{x^6} + \cancel{x^4} + \cancel{x^3} + x + x^6 + x^5 +$

$+ \cancel{x^3} + x^2 + 1 = \cancel{x^{13}} + x^{12} + x^7 + \cancel{x^6} + \cancel{x^5} + x^2 + x + 1 \;|\; x^8 + x^4 + x^3 + x + 1$

$-\cancel{x^{13}} + x^5 + \cancel{x^8} + \cancel{x^6} + \cancel{x^5}$ $\quad x^5 + x^4 + x$

$\cancel{x^{12}} + x^5 + \cancel{x^7} + \cancel{x^8} + x^2 + x + 1$

$-\cancel{x^{12}} + \cancel{x^8} + \cancel{x^7} + x^5 + x^4$

$x^8 + x^5 + x^4 + x^2 + x + 1$

$-\underline{x^8 + x^5 + x^4 + x^2 + x}$

$1 \Rightarrow A^{-1}$ is correct.

Task 3 (3 pts)

$A(x) = x^4 + x^3 + 1$; $GF(2^4)$

$e^0 = 1$

$e^1 = x = e$

$e^2 = x^2 = e^1 \cdot e \mod P$

$e^3 = x^3 = e^2 \cdot e \mod P$

$e^4 = x^3 + 1 = e^3 \cdot e \mod P$

$e^5 = x^3 + x + 1 = e^4 \cdot e \mod P$

$e^6 = x^3 + x^2 + x + 1 = e^5 \cdot e \mod P$

$e^7 = x^2 + x + 1 = e^6 \cdot e \mod P$

$e^8 = x^3 + x^2 + x = e^7 \cdot e \mod P$

$e^9 = x^2 + 1 = e^8 \cdot e \mod P$

$e^{10} = x^3 + x = e^9 \cdot e \mod P$

$e^{11} = x^3 + x^2 + 1 = e^{10} \cdot e \mod P$

$e^{12} = x + 1 = e^{11} \cdot e \mod P$

$e^{13} = x^2 + x = e^{12} \cdot e \mod P$

$e^{14} = x^3 + x^2 = e^{13} \cdot e \mod P$

$e^{15} = 1 = e^{14} \cdot e \mod P$ - final element. e^{16} begins a new cycle.

Powers of primitive elements are shown above 'e' symbol.

Task 4 (5 pts)

This task I decided to implement not only S-Box, ShiftRows and MixColumn, but the whole AES algo with key expansion.
All results are shown on a screenshot below, code in Python3.9 included.

The implemented in Python3 AES ECB 128 contains example encryption and decryption of the plaintext "**theruleroflondorheythisisshrinehandmaid**" with the key "**firelinkfirekeep**".

The **Task4** to run functions on a hex string is also implemented in the code.

All results and mid-states can be seen in the screenshot below.

```
PS G:\Horzine\Учеба\Univ\МАГА\1_Курс\ОСЕНЬ3ИМА\MFMC\HW4> python.exe .\aes.py
[*] AES128 (ECB MODE)
[*] Plaintext:  theruleroflondorheythisisshrinehandmaid
[*] Key:  firelinkfirekeep
SubByteStep_State:  [[201, 212, 1, 107], [124, 107, 118, 124], [240, 43, 114, 103], [240, 212, 103, 119]]
ShiftRowsStep_State:  [[201, 212, 1, 107], [107, 118, 124, 124], [114, 103, 240, 43], [119, 240, 212, 103]]
MixColumnsStep_State:  [[49, 190, 162, 30], [254, 97, 38, 137], [223, 103, 225, 232], [183, 141, 60, 36]]
AddRoundKeyStep_State:  [[27, 248, 130, 85], [218, 44, 2, 200], [252, 42, 222, 178], [173, 252, 40, 64]]
SubByteStep_State:  [[171, 242, 89, 119], [254, 99, 162, 43], [43, 164, 162, 99], [130, 119, 240, 173]]
ShiftRowsStep_State:  [[171, 242, 89, 119], [99, 162, 43, 254], [162, 99, 43, 164], [173, 130, 119, 240]]
MixColumnsStep_State:  [[231, 227, 147, 163], [61, 138, 5, 151], [123, 11, 189, 209], [102, 211, 5, 56]]
AddRoundKeyStep_State:  [[205, 165, 179, 232], [25, 199, 33, 214], [88, 70, 130, 139], [124, 162, 17, 92]]
SubByteStep_State:  [[197, 215, 168, 170], [197, 99, 208, 80], [71, 103, 33, 80], [48, 170, 80, 182]]
ShiftRowsStep_State:  [[197, 215, 168, 170], [99, 208, 80, 197], [33, 80, 71, 103], [182, 48, 170, 80]]
MixColumnsStep_State:  [[163, 190, 86, 44], [214, 172, 107, 194], [37, 247, 147, 81], [97, 130, 187, 231]]
AddRoundKeyStep_State:  [[137, 248, 118, 103], [242, 225, 79, 131], [6, 186, 172, 11], [123, 243, 175, 131]]
[+] B64(Encrypted):  DsKTWF/DmcO5w53DmcK/w67ChcKdLk3Dq3pOERoNwpvDgAhmLCHCncOoOUXCvgwkSnJRXBo5w7okw4DDqgEYw5HDmcOP
[+] Decrypted:  theruleroflondorheythisisshrinehandmaid
[*] Task:
Default state:  [[200, 251, 84, 49], [119, 238, 205, 229], [195, 19, 206, 240], [79, 115, 165, 238]]
SubByteStep_State:  [[232, 15, 32, 199], [245, 40, 189, 217], [46, 125, 139, 140], [132, 143, 6, 40]]
ShiftRowsStep_State:  [[232, 15, 32, 199], [40, 189, 217, 245], [139, 140, 46, 125], [40, 132, 143, 6]]
MixColumnsStep_State:  [[16, 202, 145, 234], [22, 101, 116, 183], [181, 38, 47, 194], [208, 51, 146, 214]]
Result: 0x10 0xca 0x91 0xea 0x16 0x65 0x74 0xb7 0xb5 0x26 0x2f 0xc2 0xd0 0x33 0x92 0xd6
PS G:\Horzine\Учеба\Univ\МАГА\1_Курс\ОСЕНЬ3ИМА\MFMC\HW4>
```

Code in Python3:

```
from string import *
from random import *
from base64 import *
from binascii import *

Nb = 4
Nk = 4
Nr = 10

Rcon = [
  [0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36],
  [0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00],
  [0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00],
  [0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00]
]

Sbox = [
  [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]
]

SboxInv = [
 [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]
]

def mul02(x):
 s = x << 1
 s &= 0xff
 if (x & 128) != 0:
    s = s ^ 0x1b
 return s

def mul03(x):
 return mul02(x) ^ x
```

```
def MixColumns(block):
 block_n = [[] for k in range(4)]
 for i in range(4):
     col = [block[j][i] for j in range(4)]
     col = MixColumn(col)
     for i in range(4):
        block_n[i].append(col[i])
 return block_n

def MixColumn(column):
 r = [
     mul02(column[0]) ^ mul03(column[1]) ^ column[2] ^ column[3],
     mul02(column[1]) ^ mul03(column[2]) ^ column[3] ^ column[0],
     mul02(column[2]) ^ mul03(column[3]) ^ column[0] ^ column[1],
     mul02(column[3]) ^ mul03(column[0]) ^ column[1] ^ column[2],
 ]
 return r

def SubBytes(state):
 for i in range(len(state)):
     for j in range(len(state[i])):
         x = state[i][j] >> 4
         y = state[i][j] & 15
         state[i][j] = Sbox[x][y]
 return state

def SubBytesInv(state):
 for i in range(len(state)):
     for j in range(len(state[i])):
         x = state[i][j] >> 4
         y = state[i][j] & 15
         state[i][j] = SboxInv[x][y]
 return state

def ShiftRows(state):
 cnt = 1
 for i in range(1, Nb):
     state[i] = state[i][cnt:] + state[i][:cnt]
     cnt += 1
 return state

def ShiftRowsInv(state):
 cnt = 1
 for i in range(1, Nb):
     state[i] = state[i][-cnt:] + state[i][:-cnt]
     cnt += 1
 return state

def KeyExpansion(key):
 key_arr = [ord(symbol) for symbol in key]
 if (len(key_arr) < 4 * Nk):
    for i in range(4 * Nk - len(key)):
        key_arr.append(0x01)
 key_expanded = [[] for i in range(4)]
 for r in range(4):
     for c in range(Nk):
        key_expanded[r].append(key_arr[r + 4 * c])
 for col in range(Nk, Nb * (Nr + 1)):
     if (col % Nk == 0):
         tmp = [key_expanded[row][col - 1] for row in range(1, 4)]
         tmp.append(key_expanded[0][col - 1])
```

```
        for j in range(len(tmp)):
            sb_x = tmp[j] >> 4
            sb_y = tmp[j] & 15
            tmp[j] = Sbox[sb_x][sb_y]
        for row in range(4):
            s = (key_expanded[row][col - 4]) ^ (tmp[row]) ^
(Rcon[row][int(col/Nk - 1)])
            key_expanded[row].append(s)
    else:
        for row in range(4):
            s = key_expanded[row][col - 4] ^ key_expanded[row][col - 1]
            key_expanded[row].append(s)
 return key_expanded

def AddRoundKey(state, key_expanded, rnd = 0):
 for col in range(Nk):
    s0 = state[0][col] ^ key_expanded[0][Nb * rnd + col]
    s1 = state[1][col] ^ key_expanded[1][Nb * rnd + col]
    s2 = state[2][col] ^ key_expanded[2][Nb * rnd + col]
    s3 = state[3][col] ^ key_expanded[3][Nb * rnd + col]
    state[0][col] = s0
    state[1][col] = s1
    state[2][col] = s2
    state[3][col] = s3
 return state

def SplitBlocks(data):
 res = []
 for i in range(len(data) // 16):
    b = data[i * 16: i * 16 + 16]
    matrix = [[] for l in range(4)]
    for i in range(4):
        for j in range(4):
           matrix[i].append(b[i + j * 4])
    res.append(matrix)
 return res

def AESEncrypt(plain, key):
 plain = plain + (16 - len(plain) % 16) * chr(16 - len(plain) % 16)
 plain = [ord(sym) for sym in plain]
 states = SplitBlocks(plain)
 key_expanded = KeyExpansion(key)
 temp_states = []
 checker = False
 for state in states:
   temp_states.append(AddRoundKey(state,key_expanded))
 states = temp_states
 for rnd in range(1, Nr):
    temp_states = []
    for state in states:
        state = SubBytes(state)
        if (checker == False): print('SubByteStep_State: ', state)
        state = ShiftRows(state)
        if (checker == False): print('ShiftRowsStep_State: ', state)
        state = MixColumns(state)
        if (checker == False): print('MixColumnsStep_State: ', state)
        state = AddRoundKey(state, key_expanded, rnd)
        if (checker == False): print('AddRoundKeyStep_State: ', state)
        temp_states.append(state)
    checker = True
    states = temp_states
```

```
 temp_states = []
 res = []
 for state in states:
     state = SubBytes(state)
     state = ShiftRows(state)
     state = AddRoundKey(state, key_expanded, rnd + 1)
     temp_states.append(state)
 states = temp_states
 for state in states:
     for col in range(4):
         for row in range(4):
            res.append(state[row][col])
 return res

def AESDecrypt(cipher, key):
 states = SplitBlocks(cipher)
 key_expanded = KeyExpansion(key)
 temp_states = []
 for state in states:
     state = AddRoundKey(state, key_expanded, Nr)
     state = ShiftRowsInv(state)
     state = SubBytesInv(state)
     temp_states.append(state)
 states = temp_states
 rnd = Nr - 1
 while (rnd >= 1):
     temp_states = []
     for state in states:
         state = AddRoundKey(state, key_expanded, rnd)
         for k in range(3):
            state = MixColumns(state)
         state = ShiftRowsInv(state)
         state = SubBytesInv(state)
         temp_states.append(state)
     states = temp_states.copy()
     rnd = rnd - 1
 temp_states = []
 for state in states:
    temp_states.append(AddRoundKey(state, key_expanded, rnd))
 states = temp_states
 res = []
 for state in states:
     for col in range(4):
         for row in range(4):
            res.append(state[row][col])
 return res

plaintext = 'theruleroflondorheythisisshrinehandmaid'
key = 'firelinkfirekeep'
keytmp = key
print('[*] AES128 (ECB MODE)')
print('[*] Plaintext: ', plaintext)
print('[*] Key: ', keytmp)

encrypted = AESEncrypt(plaintext, keytmp)
encrypted_out = [chr(sym) for sym in encrypted]
#print('[+] Encrypted: ', encrypted_out)
encr = ''

for p in encrypted_out:
```

```
 encr += p

print('[+] B64(Encrypted): ', b64encode(encr.encode()).decode('utf-8'))
decrypted = AESDecrypt(encrypted, key)
decrypted_out = [chr(sym) for sym in decrypted]
decr = ''

for p in decrypted_out:
 decr += p

decr = decr[:-ord(decr[len(decr)-1:])]
print('[+] Decrypted: ', decr)

print('[*] Task: ')
taskplain = unhexlify('C877C34FFBEE137354CDCEA531E5F0EE')
taskplain = taskplain + (16 - len(taskplain) % 16) * chr(16 - len(taskplain)
% 16).encode()
taskplain = [sym+0 for sym in taskplain]
states = SplitBlocks(taskplain)
state = states[0]
print('Default state: ', state)
state = SubBytes(state)
print('SubByteStep_State: ', state)
state = ShiftRows(state)
print('ShiftRowsStep_State: ', state)
state = MixColumns(state)
print('MixColumnsStep_State: ', state)
print('Result: ',end='')
for pack in state:
    for sym in pack:
        print(hex(sym), end=' ')
```