**ECE404**

**HW4**

**Yudi Wu**

**wu553@purdue.edu**

**Code:**

This code only works with python 2, probably due to wrong installation of bitvector.

#!/usr/bin/python

### ece404\_hw04\_wu.py

import sys

from BitVector import \*

def mixcol(bitvec):

AES\_modulus = BitVector(bitstring='100011011')

MUL2 = BitVector(bitstring = '00000010')

MUL3 = BitVector(bitstring = '00000011')

bitvec2 = BitVector(intVal = 0,size = 128)

for i in range(4):

bitvec2[(i\*32):(i\*32+8)] = bitvec[(i\*32):(i\*32+8)].gf\_multiply\_modular(MUL2,AES\_modulus,8) ^ \

bitvec[(i\*32+8):(i\*32+16)].gf\_multiply\_modular(MUL3,AES\_modulus,8) ^ \

bitvec[(i\*32+16):(i\*32+24)] ^ \

bitvec[(i \* 32 + 24):(i \* 32 + 32)]

bitvec2[(i\*32+8):(i\*32+16)] = bitvec[(i\*32):(i\*32+8)] ^ \

bitvec[(i\*32+8):(i\*32+16)].gf\_multiply\_modular(MUL2,AES\_modulus,8) ^ \

bitvec[(i\*32+16):(i\*32+24)].gf\_multiply\_modular(MUL3,AES\_modulus,8) ^ \

bitvec[(i\*32+24):(i\*32+32)]

bitvec2[(i\*32+16):(i\*32+24)] = bitvec[(i\*32):(i\*32+8)] ^ \

bitvec[(i\*32+8):(i\*32+16)] ^ \

bitvec[(i\*32+16):(i\*32+24)].gf\_multiply\_modular(MUL2,AES\_modulus,8) ^ \

bitvec[(i\*32+24):(i\*32+32)].gf\_multiply\_modular(MUL3,AES\_modulus,8)

bitvec2[(i\*32+24):(i\*32+32)] = bitvec[(i\*32):(i\*32+8)].gf\_multiply\_modular(MUL3,AES\_modulus,8) ^ \

bitvec[(i\*32+8):(i\*32+16)] ^ \

bitvec[(i\*32+16):(i\*32+24)]^ \

bitvec[(i\*32+24):(i\*32+32)].gf\_multiply\_modular(MUL2,AES\_modulus,8)

return bitvec2

def invmixcol(bitvec):

AES\_modulus = BitVector(bitstring='100011011')

MULE = BitVector(bitstring = '00001110')

MUL9 = BitVector(bitstring = '00001001')

MULB = BitVector(bitstring = '00001011')

MULD = BitVector(bitstring = '00001101')

bitvec2 = BitVector(intVal = 0, size = 128)

for i in range(4):

bitvec2[(i\*32):(i\*32+8)] = bitvec[(i\*32):(i\*32+8)].gf\_multiply\_modular(MULE,AES\_modulus,8) ^ \

bitvec[(i\*32+8):(i\*32+16)].gf\_multiply\_modular(MULB,AES\_modulus,8) ^ \

bitvec[(i\*32+16):(i\*32+24)].gf\_multiply\_modular(MULD,AES\_modulus,8) ^ \

bitvec[(i\*32+24):(i\*32+32)].gf\_multiply\_modular(MUL9,AES\_modulus,8)

bitvec2[(i\*32+8):(i\*32+16)] = bitvec[(i\*32):(i\*32+8)] .gf\_multiply\_modular(MUL9,AES\_modulus,8) ^ \

bitvec[(i\*32+8):(i\*32+16)].gf\_multiply\_modular(MULE,AES\_modulus,8) ^ \

bitvec[(i\*32+16):(i\*32+24)].gf\_multiply\_modular(MULB,AES\_modulus,8) ^ \

bitvec[(i\*32+24):(i\*32+32)].gf\_multiply\_modular(MULD,AES\_modulus,8)

bitvec2[(i\*32+16):(i\*32+24)] = bitvec[(i\*32):(i\*32+8)].gf\_multiply\_modular(MULD,AES\_modulus,8) ^ \

bitvec[(i\*32+8):(i\*32+16)].gf\_multiply\_modular(MUL9,AES\_modulus,8) ^ \

bitvec[(i\*32+16):(i\*32+24)].gf\_multiply\_modular(MULE,AES\_modulus,8) ^ \

bitvec[(i\*32+24):(i\*32+32)].gf\_multiply\_modular(MULB,AES\_modulus,8)

bitvec2[(i\*32+24):(i\*32+32)] = bitvec[(i\*32):(i\*32+8)].gf\_multiply\_modular(MULB,AES\_modulus,8) ^ \

bitvec[(i\*32+8):(i\*32+16)].gf\_multiply\_modular(MULD,AES\_modulus,8) ^ \

bitvec[(i\*32+16):(i\*32+24)].gf\_multiply\_modular(MUL9,AES\_modulus,8) ^ \

bitvec[(i\*32+24):(i\*32+32)].gf\_multiply\_modular(MULE,AES\_modulus,8)

return bitvec2

def getsubbyte(data,sbox):

[LE,RE] = data.divide\_into\_two()

subdata = BitVector(intVal = sbox[int(LE) \* 16 + int(RE)], size = 8)

return subdata

def gfunc(sbox,word,roundconst):

##circular shift one byte

word = word << 8

##subbyte

new\_word = getsubbyte(word[0:8],sbox) + getsubbyte(word[8:16],sbox) + getsubbyte(word[16:24],sbox) + getsubbyte(word[24:32],sbox)

##xor with roundconst

new\_word = new\_word ^ (roundconst + BitVector(intVal = 0, size = 24))

return new\_word

def getkeyschedule(subBytesTable,key):

keyschedule = []

roundconst = getRC()

for i in range(0,16,4):

keyschedule.append(BitVector(textstring =(key[i],key[i+1],key[i+2],key[i+3])))

for i in range(4,44,4):

keyschedule.append(gfunc(subBytesTable,keyschedule[i-1],roundconst[int((i-4)/4)]) ^ keyschedule[i - 4])

keyschedule.append(keyschedule[i-3] ^ keyschedule[i])

keyschedule.append(keyschedule[i-2] ^ keyschedule[i+1])

keyschedule.append(keyschedule[i-1] ^ keyschedule[i+2])

return keyschedule

def getRC():

AES\_modulus = BitVector(bitstring='100011011')

RC = BitVector(bitstring = '00000001')

roundconst = []

for i in range(10):

roundconst.append(RC)

RC = RC.gf\_multiply\_modular(BitVector(bitstring='00000010'),AES\_modulus,8)

return roundconst

def getSbox(choice):

AES\_modulus = BitVector(bitstring='100011011')

subBytesTable = [] # for encryption

invSubBytesTable = [] # for decryption

c = BitVector(bitstring='01100011')

d = BitVector(bitstring='00000101')

for i in range(0, 256):

# For the encryption SBox

a = BitVector(intVal = i, size=8).gf\_MI(AES\_modulus, 8) if i != 0 else BitVector(intVal=0)

# For byte scrambling for the encryption SBox entries:

a1,a2,a3,a4 = [a.deep\_copy() for x in range(4)]

a ^= (a1 >> 4) ^ (a2 >> 5) ^ (a3 >> 6) ^ (a4 >> 7) ^ c

subBytesTable.append(int(a))

# For the decryption Sbox:

b = BitVector(intVal = i, size=8)

# For byte scrambling for the decryption SBox entries:

b1,b2,b3 = [b.deep\_copy() for x in range(3)]

b = (b1 >> 2) ^ (b2 >> 5) ^ (b3 >> 7) ^ d

check = b.gf\_MI(AES\_modulus, 8)

b = check if isinstance(check, BitVector) else 0

invSubBytesTable.append(int(b))

if choice == 1 :

return subBytesTable

elif choice == 0:

return invSubBytesTable

def encrypt(inputfile,outputfile,key):

bv = BitVector( filename = inputfile )

FILEOUT = open(outputfile,'wb')

subBytesTable = getSbox(1) # for encryption

keyschedule = getkeyschedule(subBytesTable,key)

while(bv.more\_to\_read):

bitvec = bv.read\_bits\_from\_file(128)

bit\_ct = bitvec.length()

if(bit\_ct != 128):

bitvec.pad\_from\_right(128 - bit\_ct)

##first\_round

bitvec = bitvec ^ (keyschedule[0]+keyschedule[1]+keyschedule[2]+keyschedule[3])

for i in range(10):

##1 substition process

for j in range(16):

bitvec[j\*8:(j\*8 + 8)] = getsubbyte(bitvec[j\*8:(j\*8+8)],subBytesTable)

##2 shift row

bitvec = bitvec[0:8] + bitvec[40:48] + bitvec[80:88] + bitvec[120:128] + \

bitvec[32:40]+ bitvec[72:80] + bitvec[112:120] + bitvec[24:32] + \

bitvec[64:72]+ bitvec[104:112] + bitvec[16:24] + bitvec[56:64] + \

bitvec[96:104]+ bitvec[8:16] + bitvec[48:56] + bitvec[88:96]

##3 mix columns

if i != 9:

bitvec = mixcol(bitvec)

##4 add round key

bitvec = bitvec ^ (keyschedule[4+i\*4]+keyschedule[5 + i \*4]+keyschedule[6 + i \* 4]+keyschedule[7 + i \* 4])

bitvec.write\_to\_file(FILEOUT)

FILEOUT.close()

def decrypt(inputfile,outputfile,key):

bv = BitVector( filename = inputfile )

FILEOUT = open(outputfile,'wb')

invSubBytesTable = getSbox(0) # for decryption

keyschedule = getkeyschedule(getSbox(1),key)

while(bv.more\_to\_read):

bitvec = bv.read\_bits\_from\_file(128)

bit\_ct = bitvec.length()

if(bit\_ct != 128):

bitvec.pad\_from\_right(128 - bit\_ct)

##first\_round

bitvec = bitvec ^ (keyschedule[40]+keyschedule[41]+keyschedule[42]+keyschedule[43])

for i in range(10):

##1 inverse shift row

bitvec = bitvec[0:8] + bitvec[104:112] + bitvec[80:88] + bitvec[56:64] + \

bitvec[32:40]+ bitvec[8:16] + bitvec[112:120] + bitvec[88:96] + \

bitvec[64:72]+ bitvec[40:48] + bitvec[16:24] + bitvec[120:128] + \

bitvec[96:104]+ bitvec[72:80] + bitvec[48:56] + bitvec[24:32]

##2 substition process

for j in range(16):

bitvec[j\*8:(j\*8 + 8)] = getsubbyte(bitvec[j\*8:(j\*8+8)],invSubBytesTable)

##3 add round key

bitvec = bitvec ^ (keyschedule[36 - i\*4]+keyschedule[37 - i \*4]+keyschedule[38 - i \* 4]+keyschedule[39 - i \* 4])

##4 mix columns

if i != 9:

bitvec = invmixcol(bitvec)

bitvec.write\_to\_file(FILEOUT)

FILEOUT.close()

def main():

key = 'howtogettosesame'

while 1:

choice = raw\_input("Please choose encryption or decryption: ")

if (choice == 'encryption') | (choice == 'en') | (choice == 1):

encrypt('plaintext.txt','encryptedtext.txt',key)

break

elif (choice == 'decryption') | (choice == 'de') | (choice == 0):

decrypt('encryptedtext.txt','decryptedtext.txt',key)

break

else:

print("Please enter a valid option")

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

main()

**Output:**

yudi3160@ubuntu:/mnt/hgfs/Shared/hw4$ python ece404\_hw04\_wu.py

Please choose encryption or decryption: encryption

yudi3160@ubuntu:/mnt/hgfs/Shared/hw4$ cat encryptedtext.txt

��q���(

X��.��

jAN��{ D�˳�F��sB���L�9�

%O�ƨC�S��>}WX­�^W��\_u��v��}����c�`{��bPnU��#­W��O�p�c�Z#�ĉ?Z\*�� ��WBh�U��3��l}�9�\*�}U�ƥ�&r��Tk�0��C��m��c%x\_�F5eup�; ���=�:�>ז 4��˰g�‑b\_�4{��0dYȔY��8 ��%M��Q�

C,����.�SK`��s�L+�Uѵf�p��$.��i�'��Z~�vX��T��u`��~�7�\�t7�^�0�����]Hh����:�X��

\��e�M�)�f;"q�����\_��[���Xև ��;�F����z�~�

qK‑��D��fy��o�HM�0~R�6%-p8����[����hF���)�߭������-���t&����:���L@Y^��M:�aw�,�

^S��aNٓ�rWHq��]؜��8�:����1�

���(f�T��­|�|���4��ׄn�f<�}=���3�SG����/|�����➗$�L�C��q�� c���a�?��"

1=���֘�<{���&4�;S�J1"~-�o��H=��x�m����&���‑�^;j�4���G�r�‑0=�&��ＷҤ���˓��Ƥ�

���U�

‑���8O�����H6kǑ9�w���h'�N��b���ڔgjػ&\uEmTkə5�#g5��K��,��

a��4‑P��­಄Lf0���N��5,~07d�̳�+;�yudi3160@ubuntu:/mnt/hgfs/Shared/hw4$

yudi3160@ubuntu:/mnt/hgfs/Shared/hw4$

yudi3160@ubuntu:/mnt/hgfs/Shared/hw4$ python ece404\_hw04\_wu.py

Please choose encryption or decryption: decryption

yudi3160@ubuntu:/mnt/hgfs/Shared/hw4$ cat decryptedtext.txt

This is an unusual paragraph. I'm curious as to just how quickly you can find out what is so unusual about it. It looks so ordinary and plain that you would think nothing was wrong with it. In fact, nothing is wrong with it! It is highly unusual though. Study it and think about it, but you still may not find anything odd. But if you work at it a bit, you might find out. Try to do so without any coaching! You most probably won't, at first, find anything particularly odd or unusual or in any way dissimilar to any ordinary composition. That is not at all surprising, for it is no strain to accomplish in so short a paragraph a stunt similar to that which an author did throughout all of his book, without spoiling a good writing job, and it was no small book at that.