**Cryptography & Network Security Lab**

**PRN/ Roll No: 2019BTECS00090**

**Full name: Udaykumar Gadikar**

**Assignment No. 7**

**Title: Advanced Encryption Standard**

**Aim: To Demonstrate Advanced Encryption Standard**

**Theory:**

**AES algorithm (Rijndael algorithm) is a symmetric block cipher algorithm. The length of the data packet must be 128 bits, and the length of the key used should be 128, 192 or 256 bits. For three AES algorithms with different key lengths, they are called "AES-128", "AES-192", "AES-256".**

**Code:**

**AESencrypt**

from AESencryptfunc import \* #import AESencryptfunc module to use functions created for this program

import math #import math module to use function such as ceiling

#check that script is running with the two text files as the two parameters or else quit

if len(sys.argv) is not 3:#takes in two arguments for the plaintext.txt file name and cipherhex.txt file name

    sys.exit("Error, script needs two command-line arguments. (Plaintext.txt File and cipherhex.txt File)")

# set passphrase to be a 16 characters, 16 characters \* 8 bits = 128 bits strength

PassPhrase=""

while(len(PassPhrase)!=16):

    print("Enter in the 16 character passphrase to encrypt your text file %s" %sys.argv[1])

    PassPhrase=input()#takes in user input of char, eg. "Iwanttolearnkung"

    if(len(PassPhrase)<16):#check if less than 16 characters, if so add one space character until 16 chars

        while(len(PassPhrase)!=16):

            PassPhrase=PassPhrase+"\00"

    if(len(PassPhrase)>16):#check if bigger than 16 characters, if so then truncate it to be only 16 chars from [0:16]

        print("Your passphrase was larger than 16, truncating passphrase.")

        PassPhrase=PassPhrase[0:16]

#open plaintext.txt file to read and encrypt

file=open(sys.argv[1], "r")

message=(file.read())

print("Inside your plaintext message is:\n%s\n" % message)

file.close()

message=BitVector(textstring=message)

message=message.get\_bitvector\_in\_hex()

replacementptr=0

while(replacementptr<len(message)):

    if(message[replacementptr:replacementptr+2]=='0a'):

        message=message[0:replacementptr]+'0d'+message[replacementptr:len(message)]

        replacementptr=replacementptr+4

    else:

        replacementptr=replacementptr+2

message=BitVector(hexstring=message)

message=message.get\_bitvector\_in\_ascii()

#set up some parameters

start=0#set starting pointer for the part to encrypt of the plaintext

end=0#set ending pointer for the part to encrypt of the plaintex

length=len(message)#check the entire size of the message

loopmsg=0.00#create a decimal value

loopmsg=math.ceil(length/16)+1#use formula to figure how long the message is and how many 16 character segmentss must be encrypted

outputhex=""#setup output message in hex

#need to setup roundkeys here

PassPhrase=BitVector(textstring=PassPhrase)

roundkey1=findroundkey(PassPhrase.get\_bitvector\_in\_hex(),1)

roundkey2=findroundkey(roundkey1,2)

roundkey3=findroundkey(roundkey2,3)

roundkey4=findroundkey(roundkey3,4)

roundkey5=findroundkey(roundkey4,5)

roundkey6=findroundkey(roundkey5,6)

roundkey7=findroundkey(roundkey6,7)

roundkey8=findroundkey(roundkey7,8)

roundkey9=findroundkey(roundkey8,9)

roundkey10=findroundkey(roundkey9,10)

roundkeys=[roundkey1,roundkey2,roundkey3,roundkey4,roundkey5,roundkey6,roundkey7,roundkey8,roundkey9,roundkey10]

#set up FILEOUT to write

FILEOUT = open(sys.argv[2], 'w')

# set up the segement message loop parameters

for y in range(1, loopmsg): # loop to encrypt all segments of the message

    if(end+16<length): #if the end pointer is less than the size of the message, then set the segment to be 16 characters

        plaintextseg = message[start:end + 16]

    else: #or else if the end pointer is equal to or greator than the size of the message

        plaintextseg = message[start:length]

        for z in range(0,((end+16)-length),1): #run a while loop to pad the message segement to become 16 characters, if it is 16 already the loop will not run

            plaintextseg = plaintextseg+"\00"

            #plaintextseg2=BitVector(textstring=plaintextseg)

            #print(plaintextseg2.get\_bitvector\_in\_hex())

    #add round key zero/ find round key one

    bv1 = BitVector(textstring=plaintextseg)

    bv2 = PassPhrase

    resultbv=bv1^bv2

    myhexstring = resultbv.get\_bitvector\_in\_hex()

    for x in range(1, 10):  # loop through 9 rounds

        # sub byte

        myhexstring = resultbv.get\_bitvector\_in\_hex()

        temp1=subbyte(myhexstring)

        # shift rows

        temp2=shiftrow(temp1)

        # mix column

        bv3 = BitVector(hexstring=temp2)

        newbvashex=mixcolumn(bv3)

        newbv=BitVector(hexstring=newbvashex)

        #add roundkey for current round

        bv1 = BitVector(bitlist=newbv)

        bv2 = BitVector(hexstring=roundkeys[x-1])

        resultbv = bv1 ^ bv2

        myhexresult = resultbv.get\_bitvector\_in\_hex()

    #start round 10

    # sub byte round 10

    myhexstring = resultbv.get\_bitvector\_in\_hex()

    temp1=subbyte(myhexstring)

    # shift rows round 10

    temp2=shiftrow(temp1)

    # add round key round 10

    newbv = BitVector(hexstring=temp2)

    bv1 = BitVector(bitlist=newbv)

    bv2 = BitVector(hexstring=roundkeys[9])

    resultbv = bv1 ^ bv2

    myhexstring = resultbv.get\_bitvector\_in\_hex()

    #set encrypted hex segement of message to output string

    outputhextemp = resultbv.get\_hex\_string\_from\_bitvector()

    FILEOUT.write(outputhextemp)

    start = start + 16 #increment start pointer

    end = end + 16 #increment end pointer

# encrypted output hex string to specified cipherhex file

FILEOUT.close()

file2=open(sys.argv[2], "r")

print("The output hex value for the entire message is:\n%s\n" % file2.read())

file2.close()

**AESdecrypt**

from AESdecryptfunc import \* #import AESdecryptfunc module to use functions created for this program

import math #import math module to use function such as ceiling

import io

#check that script is running with the two text files as the two parameters or else quit

if len(sys.argv) is not 3:#takes in two arguments for the ciphertext.txt file name and plainhex.txt file name

    sys.exit("Error, script needs two command-line arguments. (Ciphertext.txt File and plainhex.txt File)")

PassPhrase=""

while(len(PassPhrase)!=16):

    print("Enter in the 16 character passphrase to decrypt your text file %s" %sys.argv[1])

    PassPhrase=input()#takes in user input of char, eg. "Iwanttolearnkung"

    if(len(PassPhrase)<16):#check if less than 16 characters, if so add one space character until 16 chars

        while(len(PassPhrase)!=16):

            PassPhrase=PassPhrase+"\00"

    if(len(PassPhrase)>16):#check if bigger than 16 characters, if so then truncate it to be only 16 chars from [0:16]

        print("Your passphrase was larger than 16, truncating passphrase.")

        PassPhrase=PassPhrase[0:16]

#open ciphertext.txt file to read and decrypt

file=open(sys.argv[1], "r")

message=(file.read())

print("Inside your ciphertext message is:\n%s\n" % message)

file.close()

#set up some parameters

start=0#set starting pointer for the part to decrypt of the ciphertext

end=32#set ending pointer for the part to decrypt of the plaintex

length=len(message)#check the entire size of the message

loopmsg=0.00#create a decimal value

loopmsg=math.ceil(length/32)+1#use formula to figure how long the message is and how many 16 character segmentss must be decrypted

outputhex=""#setup output message segment in hex

asciioutput=""#setup compilation of output message in ascii

#need to setup roundkeys here

PassPhrase=BitVector(textstring=PassPhrase)

roundkey1=findroundkey(PassPhrase.get\_bitvector\_in\_hex(),1)

roundkey2=findroundkey(roundkey1,2)

roundkey3=findroundkey(roundkey2,3)

roundkey4=findroundkey(roundkey3,4)

roundkey5=findroundkey(roundkey4,5)

roundkey6=findroundkey(roundkey5,6)

roundkey7=findroundkey(roundkey6,7)

roundkey8=findroundkey(roundkey7,8)

roundkey9=findroundkey(roundkey8,9)

roundkey10=findroundkey(roundkey9,10)

roundkeys=[roundkey1,roundkey2,roundkey3,roundkey4,roundkey5,roundkey6,roundkey7,roundkey8,roundkey9,roundkey10]

FILEOUT = io.open(sys.argv[2], 'w', encoding='utf-8')

# set up the segement message loop parameters

for y in range(1, loopmsg): # loop to encrypt all segments of the message

    plaintextseg = message[start:end]

    # add round key

    bv1 = BitVector(hexstring=plaintextseg)

    bv2 = BitVector(hexstring=roundkeys[9])

    resultbv = bv1 ^ bv2

    myhexstring = resultbv.get\_bitvector\_in\_hex()

    #inverse shift row

    myhexstring=invshiftrow(myhexstring)

    #inverse subbyte

    myhexstring=invsubbyte(myhexstring)

    for x in range(8, -1, -1):

        # add roundkey for current round

        bv1 = BitVector(hexstring=myhexstring)

        bv2 = BitVector(hexstring=roundkeys[x])

        resultbv = bv1 ^ bv2

        myhexstring = resultbv.get\_bitvector\_in\_hex()

        # mix column

        bv3 = BitVector(hexstring=myhexstring)

        myhexstring=invmixcolumn(bv3)

        # shift rows

        myhexstring = invshiftrow(myhexstring)

        # sub byte

        myhexstring = invsubbyte(myhexstring)

    #add initial round key

    bv1 = BitVector(hexstring=myhexstring)

    bv2 = PassPhrase

    resultbv = bv1 ^ bv2

    myhexstring = resultbv.get\_bitvector\_in\_hex()

    start = start + 32 #increment start pointer

    end = end + 32 #increment end pointer

    replacementptr = 0

    while (replacementptr < len(myhexstring)):

        if (myhexstring[replacementptr:replacementptr + 2] == '0d'):

            myhexstring = myhexstring[0:replacementptr] + myhexstring[replacementptr+2:len(myhexstring)]

        else:

            replacementptr = replacementptr + 2

    outputhex = BitVector(hexstring=myhexstring)

    asciioutput = outputhex.get\_bitvector\_in\_ascii()

    asciioutput=asciioutput.replace('\x00','')

    FILEOUT.write(asciioutput)

FILEOUT.close()

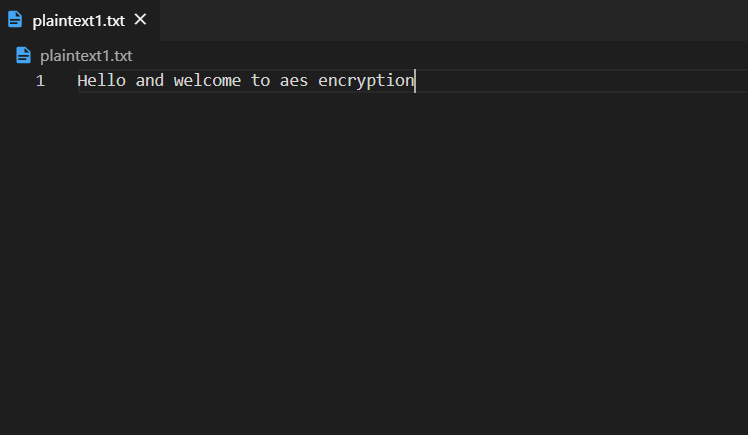
file2=io.open(sys.argv[2], "r", encoding='utf-8')

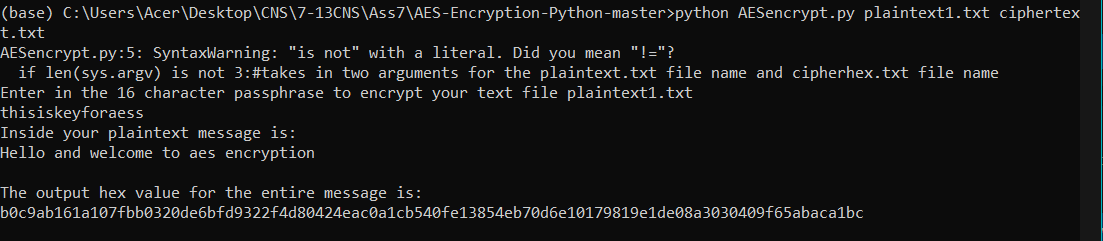
print("The decrypted message for the entire ciphertext is:\n%s\n" % file2.read())

file2.close()

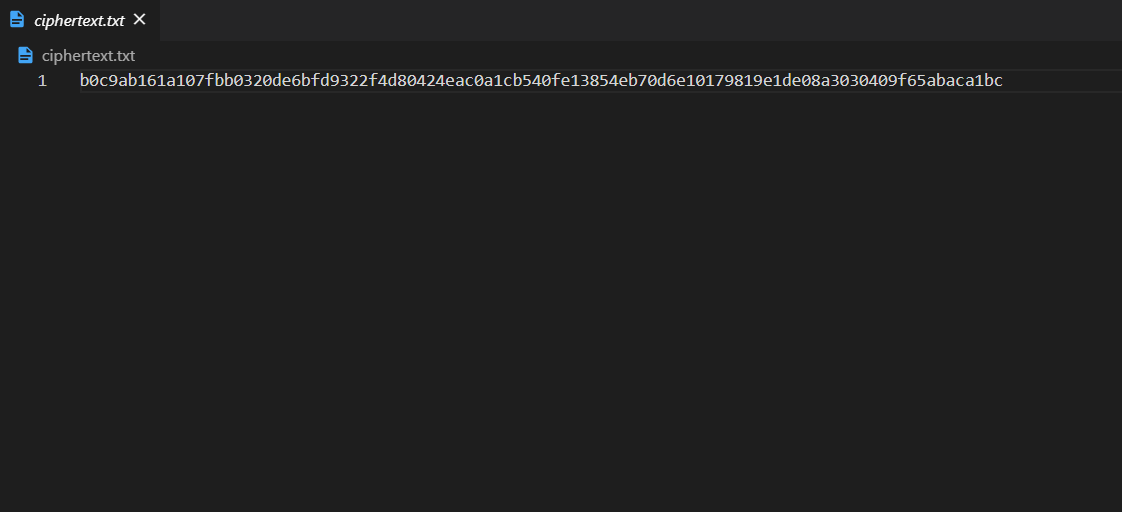
**Output:**

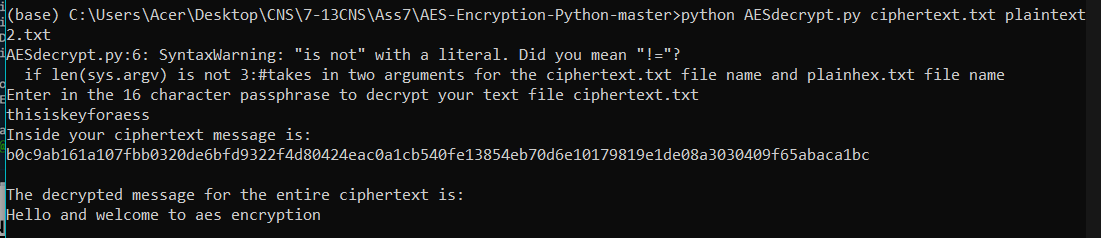
**PlainText:**

****

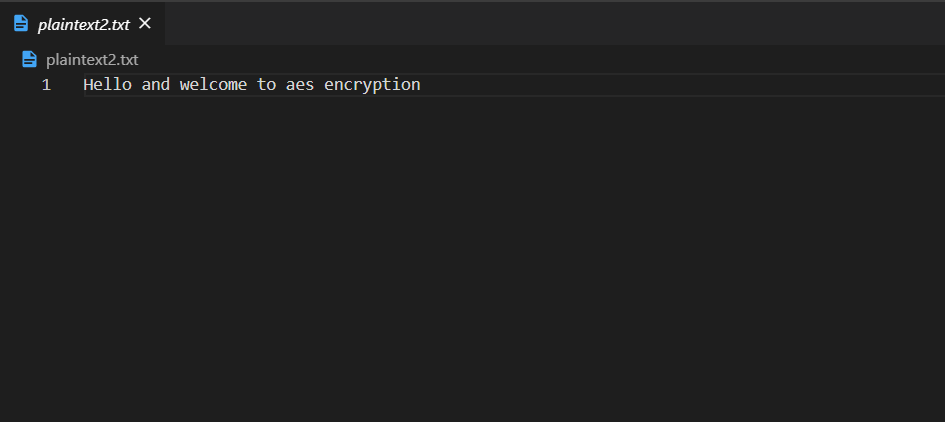
****

**Cipher Text:**

****

****

**Decrypted Plaintext2 :-**

****

**Conclusion:**

**AES instruction set is now integrated into the CPU (offers throughput of several GB/s) to improve the speed and security of applications that use AES for encryption and decryption. Even though it’s been 20 years since its introduction we have failed to break the AES algorithm as it is infeasible even with the current technology.**