Problem 1:

Encrypted output:



Begin code:

'''

Homework Number: 2

Name: Alex Goebel

ECN Login: goebel2

Due Date: 2/4/2021

'''

#!/usr/bin/env python3

### hw2\_starter.py

#All code besides main was copied from the lecture notes, some modifications were made but mostly it is the exact same as the notes

import sys

from BitVector import \*

expansion\_permutation = [31, 0, 1, 2, 3, 4, 3, 4, 5, 6, 7, 8, 7, 8, 9, 10, 11, 12, 11, 12, 13, 14, 15, 16, 15, 16, 17, 18, 19, 20, 19, 20, 21, 22, 23, 24, 23, 24, 25, 26, 27, 28, 27, 28, 29, 30, 31, 0]

pBoxPerm = [15,6,19,20,28,11,27,16,0,14,22,25,4,17,30,9,1,7,23,13,31,26,2,8,18,12,29,5,21,10,3,24]

def encrypt(inFile, encryptionKey):

key = get\_encryption\_key(encryptionKey)

round\_key = generate\_round\_keys(key)

bv = BitVector(filename = inFile)

finalString = BitVector(size = 0)

while (bv.more\_to\_read):

bitvec = bv.read\_bits\_from\_file(64)

if len(bitvec) < 64:

temp = BitVector(intVal = 0, size = 64-len(bitvec))

bitvec = bitvec + temp

if bitvec.size > 0:

for i in range(0,16):

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

newRE = RE.permute(expansion\_permutation)

out\_xor = newRE ^ round\_key[i]

RE\_modified = substitute(out\_xor)

rightHalf = RE\_modified.permute(pBoxPerm)

newRE = rightHalf ^ LE

bitvec = RE + newRE

finalString = finalString + RE + LE

return finalString

def decrypt(inFile, encryptionKey):

key = get\_encryption\_key(encryptionKey)

round\_key = generate\_round\_keys(key)[::-1]

inputFile = open(inFile)

bv = BitVector(hexstring = inputFile.read())

finalString = BitVector(size = 0)

count = 0

while (count < bv.size):

bitvec = bv[count:count+63]

if len(bitvec) <= 64:

temp = BitVector(intVal = 0, size = 64-len(bitvec))

bitvec = bitvec + temp

if bitvec.size <= count:

for i in range(0,16):

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

newRE = RE.permute(expansion\_permutation)

out\_xor = newRE ^ round\_key[i]

RE\_modified = substitute(out\_xor)

rightHalf = RE\_modified.permute(pBoxPerm)

newRE = rightHalf ^ LE

bitvec = RE + newRE

finalString = finalString + RE + LE

count += 64

final = finalString.get\_bitvector\_in\_ascii()

print(final)

return finalString

key\_permutation\_1 = [56,48,40,32,24,16,8,0,57,49,41,33,25,17,

9,1,58,50,42,34,26,18,10,2,59,51,43,35,

62,54,46,38,30,22,14,6,61,53,45,37,29,21,

13,5,60,52,44,36,28,20,12,4,27,19,11,3]

def get\_encryption\_key(encryptionKey):

key = encryptionKey

key = BitVector(textstring = key)

key = key.permute(key\_permutation\_1)

return key

key\_permutation\_2 = [13,16,10,23,0,4,2,27,14,5,20,9,22,18,11,

3,25,7,15,6,26,19,12,1,40,51,30,36,46,

54,29,39,50,44,32,47,43,48,38,55,33,52,

45,41,49,35,28,31]

shifts\_for\_round\_key\_gen = [1,1,2,2,2,2,2,2,1,2,2,2,2,2,2,1]

def generate\_round\_keys(encryption\_key):

round\_keys = []

key = encryption\_key.deep\_copy()

for round\_count in range(16):

[LKey, RKey] = key.divide\_into\_two()

shift = shifts\_for\_round\_key\_gen[round\_count]

LKey << shift

RKey << shift

key = LKey + RKey

round\_key = key.permute(key\_permutation\_2)

round\_keys.append(round\_key)

return round\_keys

s\_boxes = {i:None for i in range(8)}

s\_boxes[0] = [ [14,4,13,1,2,15,11,8,3,10,6,12,5,9,0,7],

[0,15,7,4,14,2,13,1,10,6,12,11,9,5,3,8],

[4,1,14,8,13,6,2,11,15,12,9,7,3,10,5,0],

[15,12,8,2,4,9,1,7,5,11,3,14,10,0,6,13] ]

s\_boxes[1] = [ [15,1,8,14,6,11,3,4,9,7,2,13,12,0,5,10],

[3,13,4,7,15,2,8,14,12,0,1,10,6,9,11,5],

[0,14,7,11,10,4,13,1,5,8,12,6,9,3,2,15],

[13,8,10,1,3,15,4,2,11,6,7,12,0,5,14,9] ]

s\_boxes[2] = [ [10,0,9,14,6,3,15,5,1,13,12,7,11,4,2,8],

[13,7,0,9,3,4,6,10,2,8,5,14,12,11,15,1],

[13,6,4,9,8,15,3,0,11,1,2,12,5,10,14,7],

[1,10,13,0,6,9,8,7,4,15,14,3,11,5,2,12] ]

s\_boxes[3] = [ [7,13,14,3,0,6,9,10,1,2,8,5,11,12,4,15],

[13,8,11,5,6,15,0,3,4,7,2,12,1,10,14,9],

[10,6,9,0,12,11,7,13,15,1,3,14,5,2,8,4],

[3,15,0,6,10,1,13,8,9,4,5,11,12,7,2,14] ]

s\_boxes[4] = [ [2,12,4,1,7,10,11,6,8,5,3,15,13,0,14,9],

[14,11,2,12,4,7,13,1,5,0,15,10,3,9,8,6],

[4,2,1,11,10,13,7,8,15,9,12,5,6,3,0,14],

[11,8,12,7,1,14,2,13,6,15,0,9,10,4,5,3] ]

s\_boxes[5] = [ [12,1,10,15,9,2,6,8,0,13,3,4,14,7,5,11],

[10,15,4,2,7,12,9,5,6,1,13,14,0,11,3,8],

[9,14,15,5,2,8,12,3,7,0,4,10,1,13,11,6],

[4,3,2,12,9,5,15,10,11,14,1,7,6,0,8,13] ]

s\_boxes[6] = [ [4,11,2,14,15,0,8,13,3,12,9,7,5,10,6,1],

[13,0,11,7,4,9,1,10,14,3,5,12,2,15,8,6],

[1,4,11,13,12,3,7,14,10,15,6,8,0,5,9,2],

[6,11,13,8,1,4,10,7,9,5,0,15,14,2,3,12] ]

s\_boxes[7] = [ [13,2,8,4,6,15,11,1,10,9,3,14,5,0,12,7],

[1,15,13,8,10,3,7,4,12,5,6,11,0,14,9,2],

[7,11,4,1,9,12,14,2,0,6,10,13,15,3,5,8],

[2,1,14,7,4,10,8,13,15,12,9,0,3,5,6,11] ]

def substitute( expanded\_half\_block ):

'''

This method implements the step "Substitution with 8 S-boxes" step you see inside

Feistel Function dotted box in Figure 4 of Lecture 3 notes.

'''

output = BitVector (size = 32)

segments = [expanded\_half\_block[x\*6:x\*6+6] for x in range(8)]

for sindex in range(len(segments)):

row = 2\*segments[sindex][0] + segments[sindex][-1]

column = int(segments[sindex][1:-1])

output[sindex\*4:sindex\*4+4] = BitVector(intVal = s\_boxes[sindex][row][column], size = 4)

return output

if \_\_name\_\_ == '\_\_main\_\_':

if len(sys.argv) != 5:

sys.exit('Incorrect number of arguments, please try again')

if sys.argv[1] == '-e':

inFileName = sys.argv[2]

encryptionKeyFile = sys.argv[3]

encryptionKeyFile = open(encryptionKeyFile)

encryptionKey = encryptionKeyFile.read()

output = encrypt(inFileName, encryptionKey)

outFile = sys.argv[4]

outFile = open(outFile, 'w')

outFile.write(output.get\_hex\_string\_from\_bitvector())

outFile.close()

encryptionKeyFile.close()

if sys.argv[1] == '-d':

inFileName = sys.argv[2]

encryptionKeyFile = sys.argv[3]

encryptionKeyFile = open(encryptionKeyFile)

encryptionKey = encryptionKeyFile.read()

output = decrypt(inFileName, encryptionKey)

outFile = sys.argv[4]

outFile = open(outFile, 'w')

#output.write\_to\_file(outFile)

output1 = str(output)

outFile.write(output1)

#outFile.write(output.get\_ascii\_from\_bitvector())

outFile.close()

encryptionKeyFile.close()

End code

Much of my code was used from the lecture notes. I was unable to get decrypt working fully. The final string that is returned does not seem to be correct and I was unable to find the source of the error. However, encryption seemed to work fine. I started in main by checking to make sure there was the correct amount of arguments and then checked if the user was trying to encrypt or decrypt. For decryption, I started with getting the message to encrypt and the encryption key, both of which I sent into the encrypt function. The encrypt function then called the needed functions for the keys, checked to see if padding was needing, and carried out the substitutions and XORing required for the encryption. It then performed the final switch after all 16 rounds were completed, put the two halves back together for the final string and returned it to the main function. My main function then wrote the encrypted text into an output file in hexstring format.