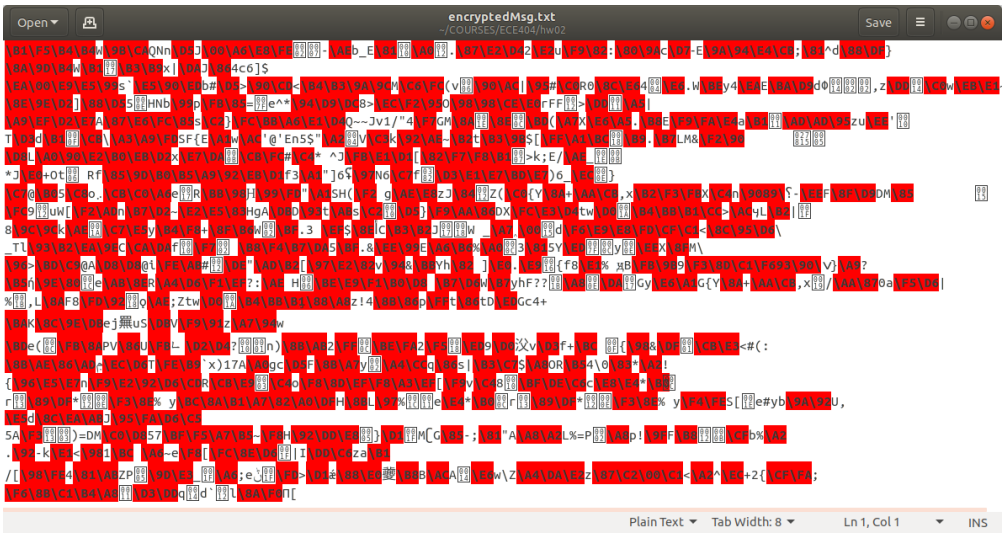
Homework Number: 02
Name: Yi Qiao
ECN Login: qiao22
Due Date: 1/24/2019

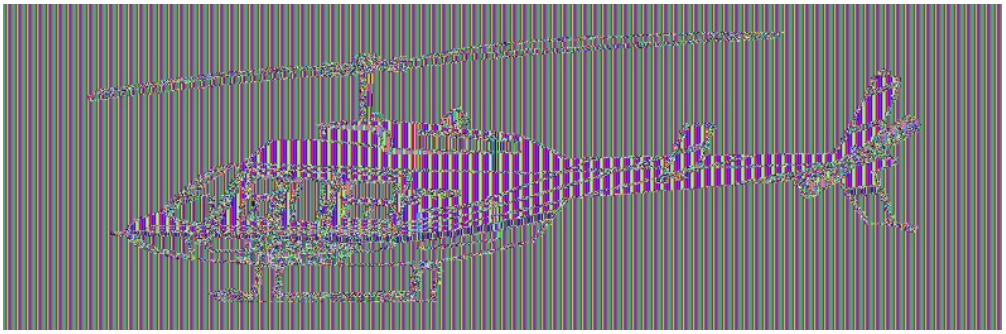
Answers

encrypted text

Since most characters is not printable, here is a screen shot with file opened with gedit.



encrypted ppm



Code

DES_text

```
#! /usr/bin/python3

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]

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]

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]

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] ]

pbox_permutation = [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 get_encrytpion_key(key):
    # return key after permutation as a bit vector
    key = BitVector(textstring=key)

    # take the first 7 bits of each byte and permute
    return key.permute(key_permutation_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


def subsitute(bv):

    # subsitute using s-boxes
```

```
    output = BitVector(size=32)
    segments = [bv[x*6:(x+1)*6] for x in range(8)]
    for sindex in range(8):
        row = 2*segments[sindex][0] + segments[sindex][-1]
        column = int(segments[sindex][1:-1])
        output[sindex*4:(sindex+1)*4] = BitVector(intVal =
        ↪  s_boxes[sindex][row][column], size=4)

    return output


def feistelFunction(rbits, roundKey):

    rbits = rbits.permute(expansion_permutation)
    rbits ^= roundKey
    rbits = subsitute(rbits)
    rbits = rbits.permute(pbox_permutation)
    return rbits

def oneRound(bv, roundKey):

    # split into two parts
    [lbits,rbits] = bv.divide_into_two()

    newLBits = rbits.deep_copy()
    newRBits = lbits ^ feistelFunction(rbits, roundKey)

    return newLBits+newRBits


def DES(msgFp, roundkey):

    output = BitVector(size=0)
    while msgFp.more_to_read:
        bv = msgFp.read_bits_from_file(64)

        # padding
        if bv.length() < 64:
            bv += BitVector(size=64-len(bv))

        # 16 round encryption
        for i in range(16):
            bv = oneRound(bv, roundKey[i])

        # swap at the end
        [lbits,rbits] = bv.divide_into_two()
        output += rbits+lbits
```

```
    return output


def encrypt(msgFp, roundkey):
    return DES(msgFp, roundkey)


def decrypt(msgFp, roundkey):
    return DES(msgFp, roundkey.reverse())


if __name__ == "__main__":

    # read key from file
    with open("key.txt", "r", encoding="UTF-8") as fp:
        key = fp.read().strip() # 8 bytes here

    key = get_encrytpion_key(key)

    print("Encrypting...")
    roundKey = generate_round_keys(key)
    msgFp = BitVector(filename="message.txt")
    encryptedMsg = encrypt(msgFp, roundKey)
    with open("encryptedMsg.txt","wb") as fp:
        encryptedMsg.write_to_file(fp)
    print("Done!")

    print("Decrypting...")
    roundKey = generate_round_keys(key)
    msgFp = BitVector(filename="encryptedMsg.txt")
    decryptedMsg = decrypt(msgFp, roundKey)

    with open("decryptedMsg.txt","wb") as fp:
        decryptedMsg.write_to_file(fp)
    print("Done!")
```

DES_image

```
#! /usr/bin/python3

import sys
from BitVector import *

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]

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]

pbox_permutation = [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]

shifts_for_round_key_gen = [1,1,2,2,2,2,2,2,1,2,2,2,2,2,2,1]

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] ]

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]

def get_encrytpion_key(key):
    key = BitVector(textstring=key)
    key = key.permute(key_permutation_1)
    return key


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


def subsitute(bv):

    output = BitVector(size=32)
    segments = [bv[x*6:(x+1)*6] for x in range(8)]
    for sindex in range(8):
        row = 2*segments[sindex][0] + segments[sindex][-1]
```

```
        column = int(segments[sindex][1:-1])
        output[sindex*4:(sindex+1)*4] = BitVector(intVal =
        ↪  s_boxes[sindex][row][column], size=4)

    return output


def feistelFunction(rbits, roundKey):

    rbits = rbits.permute(expansion_permutation)
    rbits ^= roundKey
    rbits = subsitute(rbits)
    rbits = rbits.permute(pbox_permutation)
    return rbits


def oneRound(bv, roundKey):

    # split into two parts
    [lbits,rbits] = bv.divide_into_two()

    newLBits = rbits.deep_copy()
    newRBits = lbits ^ feistelFunction(rbits, roundKey)

    return newLBits+newRBits


def ppmDESEncrypt(imgFn, keyFn):

    header = b""
    content = b""
    with open(imgFn, "rb") as fp:
        for i in range(3):
            header += fp.readline()
        content = fp.read()

    with open(keyFn, "r", encoding="UTF-8") as fp:
        key = fp.read().strip() # 8 bytes here

    key = get_encrytpion_key(key)
    roundKey = generate_round_keys(key)
    contentBv = BitVector(rawbytes=content)
    with open("image_enc.ppm","wb") as fp:
        fp.write(header)
        for i in range(0,len(contentBv),64):

                        # padding
            if i+64 > len(contentBv):
```

```
                bv = contentBv[i:]
                bv += BitVector(size=i+64-len(bv))
            else:
                bv = contentBv[i:i+64]

            for i in range(16):
                bv = oneRound(bv, roundKey[i])

            # swap at the end
            [lbits,rbits] = bv.divide_into_two()
            output = rbits+lbits
            output.write_to_file(fp)


if __name__ == "__main__":

    ppmDESEncrypt("image.ppm", "key.txt")
```