Jiaxing Yang

ECE 404

```
#!/usr/bin/env python

### hw2_starter.py

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_encryption_key(filename):

    file = open(filename)
    key = file.read()
    key = BitVector(textstring = key)
    key = key.permute(key_permutation_1)
    return key

def extract_round_key(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 encrypt(filename, round_key, file3):
    bv = BitVector( filename = filename )
    file = open(file3, "a")
    #init a
    encrypted_text =BitVector(size = 0)
    while (bv.more_to_read):
        bitvec = bv.read_bits_from_file( 64 )
        if bitvec.size > 0:
            if len(bitvec) != 64:
                # fiil the array when the bit count is not 64
                bitvec.pad_from_right(64 - len(bitvec))
            #divide the left and right bit first, then put in the boxes
```

```
            [LE, RE] = bitvec.divide_into_two()
            for i in range(0, len(round_key)):
                newRE = RE.permute( expansion_permutation )
                out_xor = newRE^(round_key[i])

                #substition with the s-box
                sub = substitute(out_xor)

                #permutation with the expansion box
                permute_RE = sub.permute(pbox_permutation)


                #now switch the right and left side, update the left side
                RE_new = permute_RE ^ LE
                LE = RE
                RE = RE_new
            encrypted_text = LE + RE
            outputhex = encrypted_text.get_hex_string_from_bitvector()
            file.write(outputhex)


    file.close()

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

def decrypt(filename, round_key, file3):
    FILEIN = open(filename)
    bv = BitVector(hexstring=FILEIN.read())  # (K)
    FILEOUT = open(file3, 'ab')  # (d)

    #bv = BitVector(filename = filename)
    #file = FILEIN.read()
    length = bv.length()
    for i in range(0, length - 64, 64):
        bitvec = bv[i:i + 64]
        if bitvec.size > 0:
            if len(bitvec) != 64:
                # fiil the array when the bit count is not 64
                bitvec.pad_from_right(64 - len(bitvec))
            [LE, RE] = bitvec.divide_into_two()
            for i in range(0, len(round_key)):
                newRE = RE.permute(expansion_permutation)
                out_xor = newRE ^ (round_key[i])
```

```
                # substition with the s-box
                sub = substitute(out_xor)

                # permutation with the expansion box
                permute_RE = sub.permute(pbox_permutation)

                # now switch the right and left side, update the left side
                RE_new = permute_RE ^ LE
                LE = RE
                RE = RE_new

            plain_text = RE + LE
            plain_text.write_to_file(FILEOUT)
    FILEOUT.close()




def main():
    type = ""
    if(len(sys.argv) == 5):
        type = sys.argv[1]
        file1 = sys.argv[2]
        key_file = sys.argv[3]
        file3 = sys.argv[4]
    else:
        file1 = sys.argv[1]
        key_file = sys.argv[2]
        file3 = sys.argv[3]

    key = get_encryption_key(key_file)

    #key = get_encryption_key("key.txt")
    round_key = extract_round_key(key)

    if type == "-e":
        encrypt(file1, round_key, file3)
    elif type == "-d":
        round_key = round_key[::-1]
        decrypt(file1, round_key, file3)
    else:
        image_encyption(file1, round_key, file3)


def image_encyption(image, round_key, output_f):
    file = open(image, "rb")
    file_out = open(output_f, "wb")

    image_info = file.readlines()
    file_out.writelines(image_info[0:3])

    file_out.close()

    image_bv = BitVector(filename = image)
```

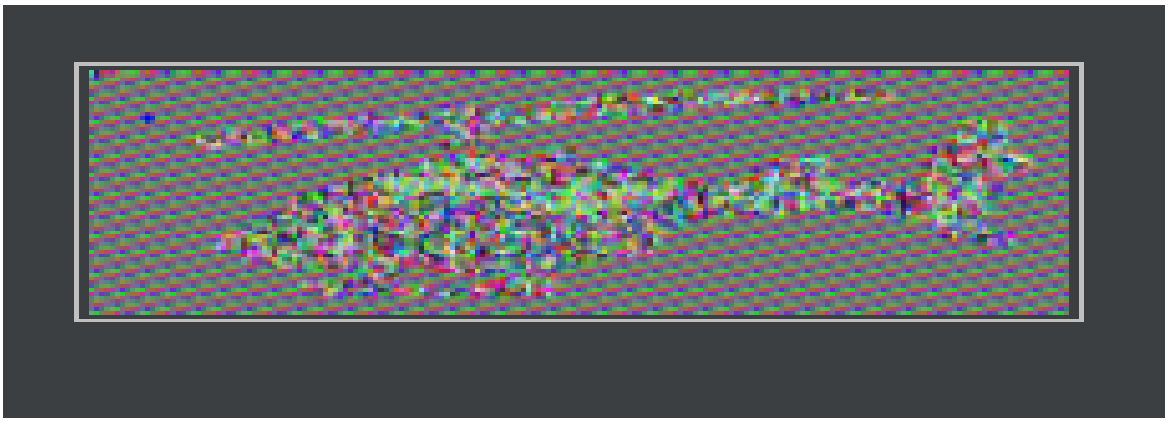
```
    image_bv.read_bits_from_file(len(image_info[0:3])*8)

    while (image_bv.more_to_read):
        bitvec = image_bv.read_bits_from_file(64)
        if bitvec.size > 0:
            if len(bitvec) != 64:
                # fiil the array when the bit count is not 64
                bitvec.pad_from_right(64 - len(bitvec))
            # divide the left and right bit first, then put in the boxes
            [LE, RE] = bitvec.divide_into_two()
            for i in range(0, len(round_key)):
                newRE = RE.permute(expansion_permutation)
                out_xor = newRE ^ (round_key[i])

                # substition with the s-box
                sub = substitute(out_xor)

                # permutation with the expansion box
                permute_RE = sub.permute(pbox_permutation)

                # now switch the right and left side, update the left side
                RE_new = permute_RE ^ LE
                LE = RE
                RE = RE_new
            encrypted_text = LE + RE
            with open(output_f, "ab") as file_out:
                encrypted_text.write_to_file(file_out)


if __name__ == '__main__':
    main()
```

The encrypted result is:

```
13083a37605c6f3e8f2254f48764e40a034a77805b6ca29b67cc02bdca45d40d453ceda544db1d8e152af
ef95d9e5465f02d82bccaeb8ec09b887e4d7ffabfe89eecc28060e21e9c05743f09b91b4f70b3d5208dca
25bad6715d10de5f20dea2e835d9edc7d59e196a7de91cc78cb6089e79e855a60d5fa4275243c30b519e0
b7df2aff211346f9197de8f64c18680dbc6d57594781405c15c5f2553ced10cecf39513673f96066c6a68
de8879ad02c416d4672966dc905602b7daf9cf82d3e42bfa27788adaa11ee0d512055b774487330187ddb
74361d8c4743d0232a0d49818eae20d3cbb6c8d1c747785d1dc3ceb0bc8dc6f5d439e252ce813e4bb0c5a
f45cbafa007b8e0daf6ae290303e6609bfe5dec49b381a819edafde688e9bb0ca9ce344abb038f76f9b64
17db0c269281dded60187c423e86ae1624b5b0c60fb93ab41f61c8d94df9936a3e126a6e846f8be41fc17
1fcaa2fe82f182ffc459b8357a0dc7e29a65126a20792057776aa23cd4e0e56cedbb681ec1b9b1269e91c
f9ff96897806d66bc309c8ab91941822c9b2f3c155642988142dabb8467b2a16fac181d83ae7c9aadaeb3
5dd0ddb66688fff8b9990314dbcabf347d731875fea81b31bcf903808a2f724a0a913363a94c2e4c7a44e
2477336636258509d13b2c6e8895bcf576770ea582374897bf9e5ad4134fd077dd67d467d996efcf0b854
89bd45b4614c352af07631e4e8357a41126719d164687f6c1a9f4bb627143c84c91548653d1c38cf1ac76
e2831085f77bc338ac7c5874b453ff64058b81e0ea1f29986c9718d10f56995cc710a9af402a1c625df23
2405910435599cb9a09aaa3e9d360bc19a55ef7bfcf6c211794baa9ec500c240e616669fab5f92618c4ce
100347b76614df1b6f16f7dea0f37c26a70de1095b1724262115942451235cd32744667bfa5fe3fb1960c
440cdfdf5cb543d7a6e4a14c1c92ebd7181d83ae7c9aadaefa57949a367c2d09426c72a5d3b80db008396
5ca76239f51895bcf576770ea58b6e290491ab1c366b508534f11554eb22c41c30f41b423a065fcf34010
```

```
0fe12fd24e65141fdf294648e5bf2046bfab2c17400312b7e8431f2cc8059f13b3aebca9b16d064dc37ef
aef06ee725f653c525b77209afb61a8375892a4aac2236d34a4158441c212665e7e02aa5222f22d77d7c3
3df7e18d44163cc43388eaf44d6ddd25bcfcb7bb16d82ec5657c78b5a5d85aef08b05d6ff40224eeb8502
85aa47bf1d3fe73f5dd6e02fdcee7aca960cd5a0551c3c0882ec8599c9a6e7e16c3c51e84aa60ba045fdd
d124601f9b16343fab0551c3c0882ec8599c9a6e7e16c3c51eb270d849f8afba5efb6f39173b506939be7
f7e550b22bf3d9ec15b07297dc61bbd238348abba294bf609c9230246930788901efd252962245777cf05
60883582f91f6550527faaff00550136ea299e9c99e8d5131600c17bdd15b65c4523fee08c6e1660c157b
0b42446af7423ff42df5d94ba8ff8d28d233972eee1d25d09d38e54310f714a2e4af60c71759f14c91835
c181e836a64cca1d60ad982dff5a437b3215271b5f587be8de2f04a5014fd5570efa46d713549f
```

The image file encryption image is:



This is the screen shot of the encrypted picture from pycharm

Problem 1 solution:

1. **Read the keys from the txt file, then use the key to generate the round keys based on the lecture code assistance.**
2. **With the round keys generated, read the plain text file with a 64-bits block each time, then divide each half to 32 bits then expand based on the given expansion permutation, then permute based on the p-box given in class.**
3. **For each round, convert the output to hexstring then write it to the file.**
4. **For the decryption part, reverse the round key list, then do the same operations but with the text = Right + left.**

Restrictions: the code only works in the environment of 3.6.4, else the output of decryption will be in random chars.

Problem 2 Solution:

1. **Read the first three lines of picture file, then based on the information, write to the encrypted picture file.**

2. **Then read the whole file in bitvector, then do the similar approach like problem1 encryption part.**