February 5, 2021 ECE 404 Prekshaa Veeraragavan
Homework 2

Problem 1:

Code:

```
#####
#Homework Number: 2
#Name: Prekshaa Veeraragavan
#ECN login: pveerar
#Due Date: February 4, 2021
######
#!/usr/bin/env python 3.7

### hw2_starter.py

import sys
import BitVector

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]

# do if -e, encrypt, if -d decrypt (go backwards through round keys.)



# !/usr/bin/env python

##  illustrate_des_substitution.py

##  Avi Kak
##  January 21, 2017

##  This is a demonstration of how you can carry out S-boxes based substitution
##  in DES.  The code shown implements the "Substitution with 8 S-boxes" step
##  that you see inside the dotted Feistel function in Figure 4 of Lecture 3 notes.

##  IMPORTANT:  This demonstration code does NOT include XORing with the round
##              key that must be carried out on the expanded right-half block
##          before it is subject to the S-boxes based substitution step
##          shown here.

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]

p_block = [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]

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
```

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

## generate_round_keys.py

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]

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



def get_encryption_key():
    inFile = sys.argv[3]
    file = open(inFile, "r")
    key = file.read()
    # key = "key.txt"
    # while True:
    #     if sys.version_info[0] == 3:
    #         key = input("\nEnter a string of 8 characters for the key: ")
    #     else:
    #         key = raw_input("\nEnter a string of 8 characters for the key: ")
    #     if len(key) != 8:
    #         print("\nKey generation needs 8 characters exactly.  Try again.\n")
    #         continue
    #     else:
    #         break
    key = BitVector(textstring=key)
    key = key.permute(key_permutation_1)
    return key

    # encryption_key = get_encryption_key()
    # round_keys = generate_round_keys(encryption_key)
    # print("\nHere are the 16 round keys:\n")
    # for round_key in round_keys:
    #     print(round_key)


def encrypt(fname):
    key = get_encryption_key()
    round_key = generate_round_keys(key)
    fileName = sys.argv[2]
    bv = BitVector(filename=fileName)
    f = open("encrypted.txt", "w")

    # i = 0
    while (bv.more_to_read):
        bitvec = bv.read_bits_from_file(64)

        if (len(bitvec) != 64):
            bitvec.pad_from_right(64-len(bitvec))
        if bitvec.length() > 0:
            print()
            #[LE, RE] = bitvec.divide_into_two()
            for i in range(0, 16):
                [LE, RE] = bitvec.divide_into_two()

                newRE = RE.permute(expansion_permutation)
                out_xor = newRE ^ (round_key[i])

                substituted = substitute(out_xor)
                permuted = substituted.permute(p_block)

                newLE = RE
                newRE = permuted ^ LE
                bitvec = newLE + newRE

                #print(bitvec.get_bitvector_in_hex())

                print(round_key[i].get_bitvector_in_hex())


            bitvec = newRE + newLE

            f.write(bitvec.get_bitvector_in_hex())


            # print(bitvec)

    f.close()



def decrypt(fname):  ## pretty much encrypt but with roundkeys opp
    key = get_encryption_key()
    round_key = generate_round_keys(key)
    fileName = sys.argv[2]
```

```
bv = BitVector(filename=fileName)
f = open("decrypted.txt", "w")
#BitVector(hexstring=f.read())
#bv = BitVector(hexstring=f.read())
#hexData = BitVector(hexstring=fileName)
f1 = open(fileName, 'r');
#BitVector(hexstring=f.read())
bv = BitVector(hexstring=f1.read())


ctr = 0
round_key.reverse()
while ctr+64 < len(bv):
    bitvec = bv[ctr:ctr + 64];
    ctr += 64
    #bitvec = bv.read_bits_from_file(64)

    if (len(bitvec) != 64):
        bitvec.pad_from_right(64 - len(bitvec))

    if bitvec.length() > 0:

        #[LE, RE] = bitvec.divide_into_two()
        for i in round_key:
        #for i in reversed(range(16))
            [LE, RE] = bitvec.divide_into_two()

            newRE = RE.permute(expansion_permutation)
            out_xor = newRE ^ (i)

            # i = i + 1
            substituted = substitute(out_xor)
            permuted = substituted.permute(p_block)

            newLE = RE
            newRE = permuted ^ LE
            bitvec = newLE + newRE
            bitvec = newRE + newLE

            f.write(bitvec.get_bitvector_in_ascii())
            #bitvec.write_to_file(f)
            #f.write(bitvec.get_text_from_bitvector())

    f.close()



if __name__ == '__main__':
    if sys.argv[1] == "-e":
        filename = sys.argv[2]
        encrypt(filename)
    elif sys.argv[1] == "-d":
        filename = sys.argv[2]
        decrypt(filename)
```

Encrypted output:

```
d04a94419ec6556c20029c83a277790c5c6380595291ecc23a40b90d60ae5b114dcefad2a37652e80dbed6bea5ab59
d92b8f043c65e1ced023bfe2aa6c4e162de19db8a75ff0f779baa3629395da7d740e784fd3150db010f0054cd9f70a13a
d6a553f954a79ca990dadc1a697ed8821548099cadedb2d6572810b06df68150cd16af08948628fab087c8577826ee1e
0ca728ea3def08044613e608e9ee27cf91a7052f2d11e6a42b371c5216e296a5486887d331794502300e42cfe9b228d
a863420c7a9d2eb3797bf08185451fc5948a61890e2fec008abe98af6a313ba886300a3041f4ca3f273f177fdd95fb97cfd
7724c196421848826c105892bfbbe47e64551e146fc6d7130d00a2dd01fa6b14a6fb6fb054f843710ddd9a311d54882d
b94802ceac4fd454332747d76b4e6be9e614545db3e6a8517c413628268c07aa64f7175ce8cd40a00a86fb279fed1361
46b2f863a0f54bb9407a21418641ba55b6641e1acb69bdf816a2cf41479f80ddde5a4a43da9f53758f152f58bb5919b65
d4a80250c259b38f498f354223cbbcbe14e5408aadc581eb0d5b19ef8219fbe42edc4e9f0467826a5c1a8141af67f0a89
7b4f212e5ab49417b576aba488381be68fc72080ee3ed00b56152e2d7da477b92c98379b694d4f466eb0d93d083fd62
d36ef1ca7f3b4399af80559ffbe0bd48b6eb441a569d479f94a54cb9ca816990971e229831db528e70972cae2f82df380
26db9db5b118ff17df3a7621911b51626ab948dd95a777b4219b0ad0ab6180def71f24b42b23444d03b974681d583e07
040d443d9365241e1fa77e1b4684da92913a6ea9a2af407d586ddf8b242706e8775ced9fa520291bbafe441dfab3c4b5
d93cfe1654202d0b7ff5c381a6a2c489e2c756eb40b6b98482a49878d04f4422fcf43605826dd6dc32cd8679e51bc800e
3ae48673c19c5890c7eec8fc58775299ea756be20afff89395ac6b021f5bb37c36e30f5948979c96b76537d8785721f1b
9789e325d2e779c4e0859c093ba756c8998219cfc497f0b7f66e259eebea3fba7a9ceed545ed833506d558c2dd9a8812
```

```
ed9bdc69e9b0bdfbd514399d2a43be6bf50a2ddab68b3c3b449a430efdd46755871a8697737a7fd251de37390186a0c
701ef7839a2b2ee99a8d6aaf540aefd111c8507120fbc296ade7e0a30846b4ad461f2af4db654e8e0008fa5a2fa42381d
8350bd431d714c42f478ca43e3f31d4e2b77c4fea7b5fc92b55c18fd29ac06b78797333758a54e7aab3439dc079d168b
7c416e23cd49084a57ff1c0974dd36102983521b30ecd7fd1931201daab5059b8139f5a3017cd7fd1931201daab744fae
27721dad95cd7fd1931201daababcce6cc5c4ddacecd7fd1931201daab462cde434ec9b646cd7fd1931201daab0013da
3321192b13b1bcd34158bc5878e3dbd126d0b7edf4cb345a27fa36e8df45ed30ec1b4cf954fd1fb2eb165e3fde33aab34
a81ef30b95855c1917ea1826f0093dfae7a9e6124ac8036677dc75ddb8cc79548b5f6b673e2aaa2e74de559d17d4c359
7eb793828ce2eba373130b87f5201f6e90c06758f
```

Decrypted output:

```
Smartphone devices from the likes of Google, LG, OnePlus, Samsung and Xiaomi are in danger of compromise by
cyber criminals after 400 vulnerable code sections were uncovered on Qualcomm's Snapdragon digital signal
processor (DSP) chip, which runs on over 40% of the global Android estate. The vulnerabilities were uncovered by
Check Point, which said that to exploit the vulnerabilities, a malicious actor would merely need to convince their target
to install a simple, benign application with no permissions at all.The vulnerabilities leave affected smartphones at risk
of being taken over and used to spy on and track their users, having malware and other malicious code installed and
hidden, and even being bricked outright, said Yaniv Balmas, Check Point's head of cyber research. Although they
have been responsibly disclosed to Qualcomm, which has acknowledged them, informed the relevant suppliers and
issued a number of alerts - CVE-2020-11201, CVE-2020-11202, CVE-2020-11206, CVE-2020-11207, CVE-2020-
11208 and CVE-2020-11209 - Balmas warned that the sheer scale of the problem could take months or even years t
```

Brief explanation of script:
Based on whether '-e' or '-d' is specified, it goes through the encrypt or decrypt function. To encrypt, a loop is used to go through the 16 rounds, each time generating a round key. Then the bitvector is divided into left half and right half. The right half passes through unchanged and becomes the new left half. Then the right half undergoes expansion permutation and is XOR with the round key. This value is substituted using the S blocks, then permuted using the p-block. This permuted value is XOR with the left half and this value becomes the new right half. The resulting bitvector is the combined left half and right half after the rounds of encryption. This is written to encrypted.txt.
The decrypt function is similar to the encrypt function, except the round keys are used in the reverse order, and input file (encrypted.txt) is in hex, so it's converted. Encryption and decryption use the same algorithm, in keeping with the properties of the Feistel Structures.