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

# Encrypt key with permutation
def get_encryption_key():
    # read key string from key.txt and turn it into a bitVector
    with open(sys.argv[3], "r") as f:
        key = f.read().strip()
    key_bv = BitVector(textstring=key)
    # extract the beginning 7 bits of each bytes and permute them
    key_bv = key_bv.permute(key_permutation_1)
    return key_bv# return the 56-bit encrypted key

# generate keys for each round
def extract_round_keys(encryption_key):
    round_keys = []
    key = encryption_key.deep_copy()
    for round_count in range(16):
        # divide the 56 relevant key bits into two 28 bit halves
        [LKey, RKey] = key.divide_into_two()
        # circularly shift to the left each half by one or two bits,
        # depending on the round
        shift = shifts_for_round_key_gen[round_count]
        LKey << shift
        RKey << shift
        key = LKey + RKey
        # apply a 56-bit to 48-bit contracting permutation
        round_key = key.permute(key_permutation_2)
        round_keys.append(round_key)
    return round_keys # resulting 48 bits constitute round keys

def substitute(newRE_xor):
    """
    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)
    # divide the right half into 8 4-bit segments
    segments = [newRE_xor[x*6:x*6+6] for x in range(8)]
    for sindex in range(len(segments)):
        # attach the last bit of the previous segment and
        # the beginning bit of the next segment to the current segments
        # the first bit and the last bit of the 6-bit segment decide the row
        row = 2*segments[sindex][0] + segments[sindex][-1]
        # the 4 bits at the mid decide the column
        column = int(segments[sindex][1:-1])
        output[sindex*4:sindex*4+4] = BitVector(intVal=s_boxes[sindex][row][column],
        size=4)
    return output

def DES(sign, fileName, round_keys):
    FILEIN = open(fileName)
    if sign == 0:
        # read plain text from message.txt
        input_bv = BitVector(textstring=FILEIN.read())
    elif sign == 1:
        # read hex text from encrypted.txt
        input_bv = BitVector(hexstring=FILEIN.read())
    # create empty bit vector to store output
    output_bv = BitVector(size=0)
    # loop through all the input and extract 64 bit at a time
    for j in range(0, input_bv.length(), 64):
        if input_bv.length() < j+64:
            # padding the last byte with 0s
            bv = input_bv[j:] + BitVector(bitlist=[0] * (j+64-input_bv.length()))
        else:
            bv = input_bv[j:j+64]
        # 16 round of 4.    Feistel Structure
        for i in range(16):
```

```python
            [LE, RE] = bv.divide_into_two()
            # expand 32-bit right-half of the input block the into 48 bits
            newRE = RE.permute(expansion_permutation)
            # key mixing: XOR with round key
            newRE_xor = newRE ^ round_keys[i]
            # S-box substitution takes the 48 bits back down to 32 bits
            newRE_sub = substitute(newRE_xor)
            # Permute the 32 bits in the order of P-box
            newRE_modified = newRE_sub.permute(pbox_permutation)
            # the new permuted right-half block XOR with the left-half block
            newRE_modified = newRE_modified ^ LE
            # concatenate the two 32-bit blocks and back into a 64-bit block
            bv = RE + newRE_modified
            # if i == 0 and j == 0:
            #     print("after:", bv.get_bitvector_in_hex())
        # switch the left-hal block and the right-half block before outputting
        [LE, RE] = bv.divide_into_two()
        output_bv += RE + LE
    return output_bv # return the bit vector of the encrypted text for the whole content


if __name__ == "__main__":
    # read key from file and encrypt the key into a 56-bit vector
    key = get_encryption_key()
    # generate 16 round keys for each round
    round_keys = extract_round_keys(key)
    # encrypt the message.txt with DES
    if sys.argv[1] == "-e":
        # perform DES encryption on the plain text
        encryptedText = DES(0,sys.argv[2], round_keys)
        # transform the ciphertext into the hex string and write out to the file
        FILEOUT = open(sys.argv[4], 'w')
        FILEOUT.write(encryptedText.get_hex_string_from_bitvector())
        FILEOUT.close()
    # decrypt the message.txt with DES
    elif sys.argv[1] == "-d":
        # perform DES decryption on the encrypted.txt with round keys in the inversed order
        decryptedText = DES(1,sys.argv[2], round_keys[::-1])
        with open(sys.argv[4], "wb") as f:
            decryptedText.write_to_file(f)

```