```
#Guanshi He
#ECE 404
#Hw 02
#Full DES implementation

### hw2_starter.py

import sys
from BitVector import *

##################################  Initial setup  ##################################

# Expansion permutation (See Section 3.3.1):
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-Box permutation (the last step of the Feistel function in Figure 4):
p_box_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]

# Initial permutation of the key (See Section 3.3.6):
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]

# Contraction permutation of the key (See Section 3.3.7):
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]

# Each integer here is the how much left-circular shift is applied
# to each half of the 56-bit key in each round (See Section 3.3.5):
shifts_key_halvs = [1,1,2,2,2,2,2,2,1,2,2,2,2,2,2,1]



##################################  S-boxes
##################################

# Now create your s-boxes as an array of arrays by reading the contents
# of the file s-box-tables.txt:
with open('s-box-tables.txt') as f:
    s_box = []
    sboxline = f.readline()
    while sboxline:
            #print sboxline
        if len(sboxline.split()) == 16:
```

```
            s_box.append([int(x) for x in sboxline.split()])
        sboxline = f.readline()


######################  Get encryptin key from user  ############################

def get_encryption_key(): # key
    ## ask user for input
    '''
    while 1:
        encrypt_key = raw_input("Enter an encryption key of at least 8 printable ASCII
characters:")
    ## make sure it satisfies any constraints on the key
        if len(encrypt_key) == 8:
            break
        else:
            print("Invalid input for encryption key")
            print("Encryption key should consist of at least 8 printable ASCII characters")
    '''
    fi = open("key.txt")
    encrypt_key = fi.readline()[:-1]

    ## next, construct a BitVector from the key
    user_key_bv = BitVector(textstring = encrypt_key)
    #print user_key_bv

    key_bv = user_key_bv.permute(key_permutation_1)        ## permute() is a BitVector function
    #print key_bv
    return key_bv


################################## Generatubg round keys
########################
def extract_round_key( nkey,i ): # round key

    [left,right] = nkey.divide_into_two()   ## divide_into_two() is a BitVector function
    left  << shifts_key_halvs[i]
    right << shifts_key_halvs[i]
    nkey = left + right
    round_key = nkey.permute(key_permutation_2)
        ##
        ##  the rest of the code
        ##
    #print round_key
    return (nkey,round_key)
```

```
########################## encryption and decryption
############################

def des(encrypt_or_decrypt, key ,bitvec):
    '''bv = BitVector( filename = input_file )
    FILEOUT = open( output_file, 'wb' )
    #bv = BitVector( filename = input_file )

    bitvec = bv.read_bits_from_file( 64 )   ## assumes that your file has an integral
                                    ## multiple of 8 bytes. If not, you must pad it.
                                    '''

    #print bitvec
    #[LE, RE] = bitvec.divide_into_two()
    #round_key = extract_round_key(key)
    #if(encrypt_or_decrypt == 'encrypt'):

    round_key = [0]*16
    for i in range(16):
        key,round_key[i] = extract_round_key(key,i)
        #print round_key[i]
    for i in range(16):
    ## write code to carry out 16 rounds of processing
        #round_key = extract_round_key(key)
        #perform the expansion permutation 32 bits to 48 bits
        [LE, RE] = bitvec.divide_into_two()
        #key,round_key = extract_round_key(key,i)
        RE_new = RE.permute(expansion_permutation)
        #print("round key = {}".format(round_key))
        #perform the xor calculation
        if(encrypt_or_decrypt == 'encrypt'):
            xor_result = RE_new ^ round_key[i]  #perform XOR when choice is encrypt
        elif(encrypt_or_decrypt == 'decrypt'):
            xor_result = RE_new ^ round_key[15 - i]  #perform XOR when choice is decrypt

        #print("xor_result = {}".format(xor_result))
        #substitution with 8 s-boxes
        RE_new2 = BitVector(size = 0)
        for j in range(8):
            #row_index = xor_result[j * 6] + xor_result[j * 6 + 5]
            row_index = BitVector(size = 2)
            row_index[0] = xor_result[j * 6]
            row_index[1] = xor_result[j * 6 + 5]
            column_index = xor_result[j*6+1:j*6+5]
            RE_new2 = RE_new2 + BitVector(intVal = (s_box[int(int(row_index) + 4*j)]
[int(column_index)]),size = 4)
        #permutation with p box
        RE_final = RE_new2.permute(p_box_permutation)
        #xor with left half
```

```
        RE_final2 = RE_final ^ LE
        #cat the left side with the new right side
        if i == 15:
            bitvec = RE_final2 + RE
        else:
            bitvec = RE + RE_final2

    #cat the encrypted text
    cipher_text = ''
    cipher_text = cipher_text + (str(bitvec))
    #print cipher_text
    #print bitvec.get_text_from_bitvector()
    cipher = bitvec.get_text_from_bitvector()
    '''fo = open(output_file,"wb")
    fo.writelines(cipher + '\n')  #write the cipher text to the output file
    fo.close()
    '''
    return bitvec



################################### main
#######################################
def main():

    ## write code that prompts the user for the key
    key = get_encryption_key()
    ## and then invokes the functionality of your implementation
    while 1:
        choice = raw_input("Please choose encrypt or decrypt: ")
        if (choice == 'encrypt') or (choice == 'decrypt'):
            break
        else:
            print("Please type in 'encrypt' or 'decrypt'")
    #implement the function DES
    fileinput = open("message.txt")
    file_plaintext = fileinput.readline()
    input_text = ''
    while file_plaintext:
        print file_plaintext
        input_text += file_plaintext
        file_plaintext = fileinput.readline()
    #print output.get_text_from_bitvector()
    # print "length",len(input_text)
    if choice == "encrypt":
        if len(input_text) % 8 == 0:
            output = BitVector(size = 0)
            #if the plaintext is 64 bytes or multiple of 64 bytes
            #just separate it into 64 bytes blocks and encrypt it
```

```
        for i in range(len(input_text) / 8):
            block = input_text[i*8:8+i*8]
            bitvec = BitVector(textstring = block)
            output += des(choice,key,bitvec)

        print output.get_text_from_bitvector()
    elif len(input_text) % 8 != 0:
        output = BitVector(size = 0)
        blocksize = len(input_text) / 8
        for i in range(blocksize):
            block = input_text[i*8:8+i*8]
            bitvec = BitVector(textstring = block)
            output += des(choice,key,bitvec)
        #perform encryption without the last block that is less than 64 bytes
        #get the last block
        #print "without last block: ",output
        block = input_text[(blocksize - 1) * 8:]
        bv1 = BitVector(textstring = block)
        temp = bv1.get_hex_string_from_bitvector()
        while len(temp) < 16:
            #pad 00 if the block is less than 64 bytes
            temp = temp + "00"
        #convert it back to bitvector
        bitvec = BitVector(hexstring = temp)
        #print "last block text : ",bitvec.get_text_from_bitvector()
        #perform des for the last block and cat it to the result
        output += des(choice,key,bitvec)
        #print output
        print output.get_text_from_bitvector()
    fo = open("encrypted.txt","wb")
    fo.writelines(output.get_text_from_bitvector())  #write the cipher text to the output file
    fo.close()
else:
    if len(input_text) % 8 == 0:
        output = BitVector(size = 0)
        for i in range(len(input_text) / 8):
            block = input_text[i*8:8+i*8]
            bitvec = BitVector(textstring = block)
            output += des(choice,key,bitvec)

        print output.get_text_from_bitvector()

    else:
        output = BitVector(size = 0)
        blocksize = len(input_text) / 8
        for i in range(blocksize - 2):
            block = input_text[i*8:8+i*8]
            bitvec = BitVector(textstring = block)
            output += des(choice,key,bitvec)
```

```
        #print "without last block:\n",output.get_text_from_bitvector()

        #get the last block of cipher text
        block = input_text[(blocksize - 1) * 8:]
        bv1 = BitVector(textstring = block)

        #print bv1.get_text_from_bitvector()

        #convert it to hex string
        temp = bv1.get_hex_string_from_bitvector()
        while len(temp) < 16:
            #pad zero to the end of the last block
            temp = temp + "00"
        #convert it back to bitvector
        bitvec = BitVector(hexstring = temp)
        #perform des decryption
        bitvec_new = des(choice,key,bitvec)

        #convert the 64 bytes last block to hex string
        temp_hex = bitvec_new.get_hex_string_from_bitvector()
        length = len(temp_hex)
        while(length > 0):
             # get rid of the padded zeros
            if(temp_hex[length - 2:length - 1] == "00"):
                length = length - 2;
            else:
                break
        #convert it back to bitvector from the correct last block
        output_new = temp_hex[0:length]
        cipher_lastblock = BitVector(hexstring = output_new)

        #print "last block:\n", cipher_lastblock.get_text_from_bitvector()
        #cat the correct last block to the result
        output = output + cipher_lastblock
        #print output.get_text_from_bitvector()

        #output the decrypted text to file
        print output.get_text_from_bitvector()
        fo = open("decrypted.txt","wb")
        fo.writelines(output.get_text_from_bitvector())
        fo.close

if __name__ == "__main__":
    main()


''' encrypted output for the given message
pal-nat184-013-060:hw02 Rio$ more message.txt
```

Shellshock, also known as Bashdoor,is a family of security bugs in the widely used Unix Bash shell, the first of which was disclosed on 24 September 2014. Many Internet-facing services, such as some web server deployments, use Bash to process certain requests, allowing an attacker to cause vulnerable versions of Bash to execute arbitrary commands. This can allow an attacker to gain unauthorized access to a computer system. Attackers exploited Shellshock within hours of the initial disclosure by creating botnets of compromised computers to perform distributed denial-of-service attacks and vulnerability scanning.Security companies recorded millions of attacks and probes related to the bug in the days following the disclosure.

```
pal-nat184-013-060:hw02 Rio$ more key.txt
sherlock
pal-nat184-013-060:hw02 Rio$ python DES_He.py
Please choose encrypt or decrypt: encrypt
Shellshock, also known as Bashdoor,is a family of security bugs in the widely used Unix Bash
shell, the first of which was disclosed on 24 September 2014. Many Internet-facing services,
such as some web server deployments, use Bash to process certain requests, allowing an
attacker to cause vulnerable versions of Bash to execute arbitrary commands. This can allow an
attacker to gain unauthorized access to a computer system. Attackers exploited Shellshock
within hours of the initial disclosure by creating botnets of compromised computers to perform
distributed denial-of-service attacks and vulnerability scanning.Security companies recorded
millions of attacks and probes related to the bug in the days following the disclosure.
??n2-?? ??@??/?^*ځ?)?xQ???sr?'rA? c?,E?|??,????Ҥj???W???g^??xG??x??HV#@??????
Ɉ?.iA??R??"Vx???B???`??
????_??G?a?W??
??a???2??w)?h???\?
?Xl'!?Z4ʒH?2??U]_?HA?40??x?w?????c???:?\8?a^G?Ա97F??_?w?+H????  ?
                                          ??=Q'???79u???Z?)???!?1WbAC?*?z??tk?м??F9e??
$~?IK?b?TR<?c???}?ѦU???i??4G?^??????}??YQzAMu3#?pi??4"?"???Ml?j???Q
                                        ?N????T?{?曜}0:?X??dx??}?HW`TnS،>??P?/?%P"]!
j???]u??A?H?eUj?^?]??]??
          ???6?ҹ®4??*?????[=_ggly?????$??!?>Y7?R??bwB*?????????
                                                ?'???
                                                   +Ð?.?&s??2?Yx?@6Z?P???J??9)?Wʆ!?z?9?
o??'???"??S{???f[???O?ZF"?,q???d˝E??G?j??=?M??TZ?γo?Tѫ???2HKI?>z?*I???&;]!?-?^4?
K?7l?t   b??N???????6x??R?EЁ
pal-nat184-013-060:hw02 Rio$ cp encrypted.txt message.txt
pal-nat184-013-060:hw02 Rio$ python DES_He.py
Please choose encrypt or decrypt: decrypt
??n2-?? ??@??/?^*ځ?)?xQ???sr?'rA? c?,E?|??,????Ҥj???W???g^??xG??x??HV#@??????
Ɉ?.iA??R??"Vx???B???`??

????_??G?a?W??

??a???2??w)?h???\?
?Xl'!?Z4ʒH?2??U]_?HA?40??x?w?????c???:?\8?a^G?Ա97F??_?w?+H????  ?
                                          ??=Q'???79u???Z?)???!?1WbAC?*?z??tk?м??F9e??
$~?IK?b?TR<?c???}?ѦU???i??4G?^??????}??YQzAMu3#?pi??4"?"???Ml?j???Q
                                        ?N????T?{?曜}0:?X??dx??}?HW`TnS،>??P?/?%P"]!
j???]u??A?H?eUj?^?]??]??
```

???6?ᵸᵸ®4??*?????[=_ggly?????$??!?>Y7?R??bwB*?????????
?'???
+Ð?.?&s??2?Yx?@6Z?P???J??9)?Wƒ!?z?9?
o??'???"??S{???f[???O?ZF"?,q???d˝E??Ɠ?j??˝?M??TZ?ɣo?T₹???2HKI?>z?*I???&;]!?-?^4?
K?7l?t b??N???????6x??R?EË
Shellshock, also known as Bashdoor,is a family of security bugs in the widely used Unix Bash shell, the first of which was disclosed on 24 September 2014. Many Internet-facing services, such as some web server deployments, use Bash to process certain requests, allowing an attacker to cause vulnerable versions of Bash to execute arbitrary commands. This can allow an attacker to gain unauthorized access to a computer system. Attackers exploited Shellshock within hours of the initial disclosure by creating botnets of compromised computers to perform distributed denial-of-service attacks and vulnerability scanning.Security companies recorded millions of attacks and probes related to the bug in the days following the disclosure.

'''

```
import sys
import random
from BitVector import *
from DES_He_simplified import *

#get the encryption key
key = get_encryption_key()
#write the original 64 bits test file
plaintext = BitVector(size = 64)
plaintext = plaintext.gen_rand_bits_for_prime(64)
plaintext_char = plaintext.get_text_from_bitvector()
inputfile = open("plaintext.txt","wb")
inputfile.writelines(plaintext_char)
inputfile.close()
print plaintext
print plaintext_char

#get the original cipher text
choice = "encrypt"

plaintext_new = [0] * 64
ciphertext = des(choice,"plaintext.txt","outputtxt.txt",key)
#change each bits of plaintext for 64 times

for i in range(64):
        plaintext_temp = BitVector(size = 64)
        plaintext_temp = plaintext
        #plaintext_temp[i] = ~plaintext[i]
        if plaintext_temp[i] == 1:
                plaintext_temp[i] = 0
        else:
                plaintext_temp[i] = 1
        plaintext_new[i] = plaintext_temp
        #print plaintext_new[i]

#des operation for 64 different cases
ciphertext_new = [0] * 64
for i in range(64):
        inputfile = open("plaintext_temp.txt","wb")
        plaintext_new_char = plaintext_new[i].get_text_from_bitvector()
        inputfile.writelines(plaintext_new_char)
        inputfile.close()
        ciphertext_new[i] = des(choice,"plaintext_temp.txt","output_new.txt",key)

#count the bit changed for 64 cases
bits_changed_count = 0

for i in range (64):
```

```
        xor_bit = ciphertext ^ ciphertext_new[i]
        bits_changed_count += xor_bit.count_bits()

average_changed = bits_changed_count / 64
print "Average bits changed for Diffusion = {}".format(average_changed)

################################################################################
#############

#result for hw02 problem2 question 1
#########Average bits changed for Diffusion = 35



#generate s-box randomly
fo = open("sbox_new.txt","wb")
for i in range(8):
        fo.writelines("S" + str(i) + ":" + "\n\n")
        for j in range(4):
                row = [] * 16
                row = random.sample(range(16),16)
                for k in range(16):
                        fo.writelines(str(row[k]) + " ")
                fo.write("\n")
        fo.write("\n")

fo.close()

#perform the same operation like the first problem
#change the s box file in DES_He.py and run Average_He.py again

#result for hw02 problem2 question 2
###########Average bits changed for Diffusion = 38

#######change one bit of encryption key

key_new = BitVector(size = 64)
key_new = key_new.gen_rand_bits_for_prime(64)
#key_new_char = key_new.get_text_from_bitvector()

key_new2 = [0] * 64
#perform des operation and get the original cipher text
original_cipher = des(choice,"input.txt","output_new.txt",key_new)

#perform change one bit of encryption key for 64 times
for i in range(64):
        key_temp = BitVector(size = 64)
        key_temp = key_new
```

```
        if key_new[i] == 1:
                key_temp[i] = 0
        else:
                key_temp[i] = 1
        key_new2[i] = key_temp

#perform DES encryption for 64 keys
cipher_new = [0] *64
for i in range(64):
        cipher_new[i] = des(choice,"plaintext_temp.txt","output_new.txt",key_new2[i])

#count the bit changed for 64 cases
bits_changed_count = 0
for i in range (64):
        xor_bit = original_cipher ^ cipher_new[i]
        bits_changed_count += xor_bit.count_bits()

average_changed = bits_changed_count / 64
print "Average bits changed for Confussion = {}".format(average_changed)

#result for hw02 problem2 question 3
########Average bits changed for Confussion = 33




'''DES_He_simplified
#Guanshi He
#ECE 404
#Hw 02
#Full DES implementation

### hw2_starter.py

import sys
from BitVector import *

#################################  Initial setup  ##################################

# Expansion permutation (See Section 3.3.1):
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-Box permutation (the last step of the Feistel function in Figure 4):
p_box_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]
```

```
# Initial permutation of the key (See Section 3.3.6):
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]

# Contraction permutation of the key (See Section 3.3.7):
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]

# Each integer here is the how much left-circular shift is applied
# to each half of the 56-bit key in each round (See Section 3.3.5):
shifts_key_halvs = [1,1,2,2,2,2,2,2,1,2,2,2,2,2,2,1]




##################################   S-boxes
##################################

# Now create your s-boxes as an array of arrays by reading the contents
# of the file s-box-tables.txt:
with open('sbox_new.txt') as f:
    s_box = []
    sboxline = f.readline()
    while sboxline:
            #print sboxline
        if len(sboxline.split()) == 16:
            s_box.append([int(x) for x in sboxline.split()])
        sboxline = f.readline()




######################  Get encryptin key from user  ############################

def get_encryption_key(): # key
    ## ask user for input
    while 1:
        encrypt_key = raw_input("Enter an encryption key of at least 8 printable ASCII
characters:")
    ## make sure it satisfies any constraints on the key
        if len(encrypt_key) == 8:
            break
        else:
            print("Invalid input for encryption key")
            print("Encryption key should consist of at least 8 printable ASCII characters")
```

```
    ## next, construct a BitVector from the key
    user_key_bv = BitVector(textstring = encrypt_key)
    #print user_key_bv

    key_bv = user_key_bv.permute(key_permutation_1)        ## permute() is a BitVector function
    #print key_bv
    return key_bv


################################## Generatubg round keys
########################
def extract_round_key( nkey,i ): # round key

    [left,right] = nkey.divide_into_two()   ## divide_into_two() is a BitVector function
    left  << shifts_key_halvs[i]
    right << shifts_key_halvs[i]
    nkey = left + right
    round_key = nkey.permute(key_permutation_2)
        ##
        ##  the rest of the code
        ##
    #print round_key
    return (nkey,round_key)



########################### encryption and decryption
############################

def des(encrypt_or_decrypt, input_file, output_file, key ):
    bv = BitVector( filename = input_file )
    FILEOUT = open( output_file, 'wb' )
    #bv = BitVector( filename = input_file )
    bitvec = bv.read_bits_from_file( 64 )   ## assumes that your file has an integral
                                    ## multiple of 8 bytes. If not, you must pad it.
    #print bitvec
    #[LE, RE] = bitvec.divide_into_two()
    #round_key = extract_round_key(key)
    #if(encrypt_or_decrypt == 'encrypt'):
    round_key = [0]*16
    for i in range(16):
        key,round_key[i] = extract_round_key(key,i)
        #print round_key[i]
    for i in range(16):
    ## write code to carry out 16 rounds of processing
        #round_key = extract_round_key(key)
        #perform the expansion permutation 32 bits to 48 bits
        [LE, RE] = bitvec.divide_into_two()
```

```
        #key,round_key = extract_round_key(key,i)
        RE_new = RE.permute(expansion_permutation)
        #print("round key = {}".format(round_key))
        #perform the xor calculation
        if(encrypt_or_decrypt == 'encrypt'):
            xor_result = RE_new ^ round_key[i]  #perform XOR when choice is encrypt
        elif(encrypt_or_decrypt == 'decrypt'):
            xor_result = RE_new ^ round_key[15 - i]  #perform XOR when choice is decrypt

        #print("xor_result = {}".format(xor_result))
        #substitution with 8 s-boxes
        RE_new2 = BitVector(size = 0)
        for j in range(8):
            #row_index = xor_result[j * 6] + xor_result[j * 6 + 5]
            row_index = BitVector(size = 2)
            row_index[0] = xor_result[j * 6]
            row_index[1] = xor_result[j * 6 + 5]
            column_index = xor_result[j*6+1:j*6+5]
            RE_new2 = RE_new2 + BitVector(intVal = (s_box[int(int(row_index) + 4*j)]
[int(column_index)]),size = 4)
        #permutation with p box
        RE_final = RE_new2.permute(p_box_permutation)
        #xor with left half
        RE_final2 = RE_final ^ LE
        #cat the left side with the new right side
        if i == 15:
            bitvec = RE_final2 + RE
        else:
            bitvec = RE + RE_final2

    #cat the encrypted text
    cipher_text = ''
    cipher_text = cipher_text + (str(bitvec))
    #print cipher_text
    #print bitvec.get_text_from_bitvector()
    cipher = bitvec.get_text_from_bitvector()
    fo = open(output_file,"wb")
    fo.writelines(cipher + '\n')  #write the cipher text to the output file
    fo.close()
    return bitvec



##################################### main
#########################################
def main():

    ## write code that prompts the user for the key
    key = get_encryption_key()
```

```
    ## and then invokes the functionality of your implementation
    while 1:
        choice = raw_input("Please choose encrypt or decrypt: ")
        if (choice == 'encrypt') or (choice == 'decrypt'):
            break
        else:
            print("Please type in 'encrypt' or 'decrypt'")
    #implement the function DES
    output = des(choice,'input.txt','output.txt',key)

if __name__ == "__main__":
    main()

'''
```