Homework Number: hw02
Name: Shu Hwai Teoh
ECN Login: teoh0
Due Date: Thursday 01/30/2020 at 4:29PM

1. Problem 1
    I. Explanation:
        i. To encrypt message.txt as encrypted.txt, I did the following steps:
            1. Read key.txt and encrypt the 64-bit key into a 56-bit vector by extracting the beginning 7 bits of each bytes and permute them with “key_permutation_1”.
            2. Generate 16 round keys for each round:
                A. Divide the 56 relevant key bits into two 28 bit halves
                B. Circularly shift to the left each half by one or two bits depending on the round with “shifts_for_round_key_gen”.
                C. Apply a 56-bit to 48-bit contracting permutation with “key_permutation_2”.
            3. encrypt the message.txt with DES
                A. Read plain text from message.txt as a bit vector
                B. Loop through all the input, extract 64 bits at a time as a block (pad the last block with 0s if the length of the last block is less than 64 bits) and Encrypted each block with 16 rounds of the following steps
                    i. Divide each block as a 32-bit left half and a 32-bit right half
                    ii. Expand 32-bit right-half into 48 bits
                    iii. XOR the 48-bit right half with round key as the new 48-bit right half
                    iv. Perform S-box substitution to let the new 48-bit right half back down to 32 bits as the 32-bit modified right half
                    v. Permute the 32-bit modified right half in the order of P-box as the 32-bit permuted right half
                    vi. XOR the 32-bit permuted right half with the original 32-bit left-half block as the final 32-bit right half
                    vii. Concatenate the original 32-bit left-half with the final 32-bit right half as the 64-bit input of the next round
                C. Switch the left-half block and the right-half block of the final

output from the previous steps

4. Concatenate all the encrypted output and write it out to encrypted.txt in hex string.

ii. To decrypt encrypted.txt as decrypted.txt, I used the same process as encryption instead that I used the 16 round keys in reversed order.

II. Encrypted output for the text

605c6f3e13083a378764e40a8f2254f45b6ca29b034a7780ca45d40d67cc02
bd44db1d8e453ceda55d9e5465152afef9caeb8ec0f02d82bc7ffabfe89b887
e4d60e21e9c9eecc280b91b4f7005743f09ca25bad6b3d5208d5f20dea2715
d10dec7d59e19e835d9edc78cb6086a7de91ca60d5fa49e79e8550b519e0b
275243c311346f917df2aff2c18680db97de8f64781405c1c6d57594ced10ce
c5c5f25533f96066cf395136779ad02c46a68de8866dc905616d46729cf82d
3e402b7daf98adaa11e2bfa27785b774487e0d51205b74361d8330187dd32
a0d498c4743d023cbb6c8d18eae20dd1dc3ceb1c7477855d439e250bc8dc6
fbb0c5af42ce813e47b8e0daf5cbafa003e6609bf6ae29030381a819ee5dec4
9be9bb0ca9dafde688038f76f9ce344abbc269281db6417db0c423e86aded6
01870c60fb93e1624b5b8d94df99ab41f61ca6e846f836a3e1261fcaa2febe4
1fc17c459b83582f182ff9a65126a7a0dc7e2776aa23c20792057edbb681ed
4e0e56c9e91cf9fc1b9b1266d66bc30f968978041822c9b9c8ab9194298814
22f3c1556b2a16facdabb84677c9aadae181d83aeb66688ffb35dd0dd14dbc
abff8b9990375fea81b347d7318808a2f7231bcf90363a94c2e4a0a9133477
336634c7a44e213b2c6e86258509d6770ea58895bcf57f9e5ad412374897b
d67d467d34fd077db85489bd996efcf0352af07645b4614c7a41126731e4e8
357f6c1a9f19d164683c84c9154bb6271438cf1ac748653d1c5f77bc336e283
1084b453ff68ac7c5870ea1f2994058b81e10f5699586c9718df402a1c6cc71
0a9a0591043525df23249aaa3e9d599cb9a055ef7bfc360bc19a4baa9ec5f6c
2117916669fab00c240e64ce100345f92618cf1b6f16f7b76614dc26a70de7d
ea0f37426211591095b172cd327446424512353fb1960c67bfa5fe5cb543d7
440cdfdf1c92ebd7a6e4a14c7c9aadae181d83ae367c2d09fa57949ad3b80d
b0426c72a576239f51083965ca6770ea58895bcf571ab1c366b6e29049115
54eb2b508534f41b423a02c41c30f100fe12f65fcf3401fdf2946d24e651446b
fab2c48e5bf20b7e8431f1740031213b3aebc2cc8059f4dc37efaa9b16d065f
653c52ef06ee72fb61a8375b77209ac2236d345892a4aac212665ea415844
122f22d777e02aa52e18d4416d7c33df7eaf44d6d3cc43388b7bb16d8dd25
bcfc78b5a5d82ec5657c5d6ff4025aef08b0285aa47b24eeb850f5dd6e02f1d
3fe73a960cd5afdcee7ac882ec8590551c3c016c3c51e9c9a6e7e045fddd184
aa60ba16343fab24601f9b882ec8590551c3c016c3c51e9c9a6e7ef8afba5eb
270d8493b506939fb6f39170b22bf3dbe7f7e55297dc61b9ec15b07abba294

bbd23834802469307f609c9232529622488901efd608835825777cf05527fa
afff91f6550ea299e9c005501361600c17b99e8d5134523fee0dd15b65cc157
b0b48c6e166023ff42df2446af74f8d28d235d94ba8fd25d09d33972eee171
4a2e4a8e54310f9f14c918f60c717536a64cca35c181e82dff5a431d60ad981
b5f587b7b321527a5014fd5e8de2f04d713549f570efa46

III. Decrypted output for the text

Earlier this week, security researchers took note of a series of changes Linux and Windows developers began rolling out in beta updates to address a critical security flaw: A bug in Intel chips allows low-privilege processes to access memory in the computer's kernel, the machine's most privileged inner sanctum. Theoretical attacks that exploit that bug, based on quirks in features Intel has implemented for faster processing, could allow malicious software to spy deeply into other processes and data on the target computer or smartphone. And on multi-user machines, like the servers run by Google Cloud Services or Amazon Web Services, they could even allow hackers to break out of one user's process, and instead snoop on other processes running on the same shared server. On Wednesday evening, a large team of researchers at Google's Project Zero, universities including the Graz University of Technology, the University of Pennsylvania, the University of Adelaide in Australia, and security companies including Cyberus and Rambus together released the full details of two attacks based on that flaw, which they call Meltdown and Spectre.

IV. Script

```
#!/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] ]
```

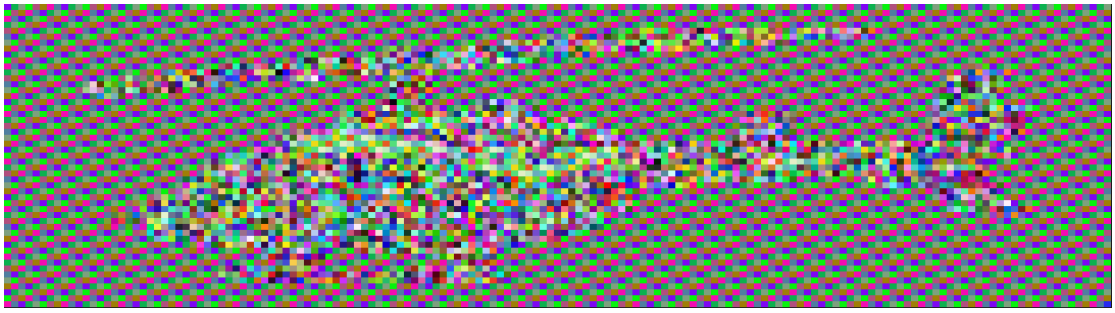
```
s_boxes[4] = [ [2,12,4,1,7,10,11,6,8,5,3,15,13,0,14,9],
                        [14,11,2,12,4,7,13,1,5,0,15,10,3,9,8,6],
                        [4,2,1,11,10,13,7,8,15,9,12,5,6,3,0,14],
                        [11,8,12,7,1,14,2,13,6,15,0,9,10,4,5,3] ]

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):
                [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)
```

2. Problem 2
    I. Explanation: to encrypt image.ppm as image_enc.ppm, I did the following steps:
        i. Read key.txt and encrypt the 64-bit key into a 56-bit vector like the process described in problem 1.
        ii. Generate 16 round keys for each round like the process described in problem 1.
        iii. Read the first three lines in image.ppm as the header
        iv. encrypt the rest of the data in image.ppm with DES like the process described in problem 1
        v. Write the original header to image_enc.ppm.
        vi. Concatenate all the encrypted output and write it out to image_enc.ppm.
    II. Picture of the encrypted PPM image



III. Script

```
#!/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[2], "r", encoding="UTF-8") 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)
    segments = [newRE_xor[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 DES(data, round_keys):
    input_bv = BitVector(rawbytes=data)
    output_bv = BitVector(size=0)
    for j in range(0, input_bv.length(), 64):
        if input_bv.length() < j+64:
            bv = input_bv[j:] + BitVector(bitlist=[0] * (j+64-input_bv.length()))
        else:
            bv = input_bv[j:j+64]
        for i in range(16):
            [LE, RE] = bv.divide_into_two()
            # expand the 32-bit block into 48 bits
            newRE = RE.permute(expansion_permutation)
            # 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)
            # P-box
            newRE_modified = newRE_sub.permute(pbox_permutation)
            newRE_modified = LE ^ newRE_modified
            bv = RE + newRE_modified
        [LE, RE] = bv.divide_into_two()
        output_bv += RE + LE
    return output_bv
```

```
if __name__ == "__main__":
    # DES_image.py image.ppm key.txt image_enc.ppm
    # 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)
    # initialize a variable to store the header
    header = b""
    with open(sys.argv[1], "rb") as f:
        for i in range(3):
            # read the header from image.ppm
            header += f.readline()
        # read other data from image.ppm
        data = f.read()
    # encrypt the image.ppm with DES
    encryptedImg = DES(data, round_keys)
    with open(sys.argv[3], "wb") as f:
        f.write(header)
        encryptedImg.write_to_file(f)
```