ECE404 HW2

Yudi Wu

0026402011

wu553@purdue.edu

Problem 1

My solution only works with Python2, and it takes ‘message.txt’ as input for encryption or decryption, and it output ‘encrypted.txt’ or ‘decrypted.txt’ depends on your choice.

Code

#!/usr/bin/env/python

### DES\_wu.py

##This program only works well with Python2, running with Python3 would result in unknown bugs

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:

arrays = []

with open('s-box-tables.txt') as f:

for line in f:

##store lines wih length larger than 4 into arrays

if len(line) > 4:

row = line.split()

arrays.append(row)

s\_box = []

##Seperate arrays into 8 boxes

for i in range(0,32, 4):

s\_box.append([arrays[k] for k in range(i, i+4)]) # S\_BOX

f.close()

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

def get\_encryption\_key(): # key

## ask user for input

while True:

key = raw\_input("Please enter a 8 characters key: ")

## make sure it satisfies any constraints on the key

if len(key) == 8:

break

else:

print("Your key is not 8 characters long")

user\_key\_bv = BitVector(textstring = key)

key\_bv = user\_key\_bv.permute(key\_permutation\_1) ## permute() is a BitVector function

return key\_bv

################################# Generatubg round keys ########################

def extract\_round\_key( nkey ): # round key

round\_key = []

for i in range(16):

[left,right] = nkey.divide\_into\_two() ## divide\_into\_two() is a BitVector function

##Shift two halves

left << shifts\_key\_halvs[i]

right << shifts\_key\_halvs[i]

##Combine two halves and permute

nkey = left + right

nkey.permute(key\_permutation\_2)

round\_key.append(nkey)

return round\_key

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

def des(encrypt\_or\_decrypt, input\_file, key ):

bv = BitVector( filename = input\_file )

##change output\_file name depends on Encrypt or Decrypt mode

if encrypt\_or\_decrypt == 1:

output\_file = 'encrypted.txt'

else:

output\_file = 'decrypted.txt'

##Set write byte mode

FILEOUT = open( output\_file, 'wb' )

while(bv.more\_to\_read):

bitvec = bv.read\_bits\_from\_file( 64 ) ## assumes that your file has an integral

##find if bitvec is complete 64 bit, if no, padding from the right

bit\_ct = bitvec.length()

if(bit\_ct != 64):

bitvec.pad\_from\_right(64 - bit\_ct)

[LE, RE] = bitvec.divide\_into\_two()

##extract all round keys

round\_key = extract\_round\_key(key)

for i in range(16):

##E-step

temp3 = RE

temp = RE.permute(expansion\_permutation)

if(encrypt\_or\_decrypt == 1):

temp = temp ^ round\_key[i]

else:

temp = temp ^ round\_key[15 - i]

##S-BOX

temp2 = BitVector(size = 0)

for i in range(8):

row = 2 \* temp[i \* 6] + temp[i \* 6 + 5]

col = 8 \* temp[i \* 6 + 1] + 4 \* temp[i \* 6 + 2] + 2 \* temp[i \* 6 + 3] + temp[i \* 6 + 4]

temp2 += BitVector(intVal = int(s\_box[i][row][col]), size = 4)

##P-BOXAAAAAAAAAAAAAAAA

temp2 = temp2.permute(p\_box\_permutation)

RE = LE ^ temp2

LE = temp3

##output

combined = RE + LE

combined.write\_to\_file(FILEOUT)

FILEOUT.close()

#################################### main #######################################

def main():

## write code that prompts the user for the key

## and then invokes the functionality of your implementation

key = get\_encryption\_key()

while True:

function = raw\_input("Encryption or Decryption?: ")

if (function == 'encryption') or (function == 'Encryption') or (function == 'En') or (function == 'en'):

function = 1

break

elif (function == 'decryption') or (function == 'Decryption') or (function == 'De') or (function == 'de'):

function = 0

break

else:

print("Please enter a valid option.\n")

des(function,’message.txt’,key)

if \_\_name\_\_ == "\_\_main\_\_":

main()

Output

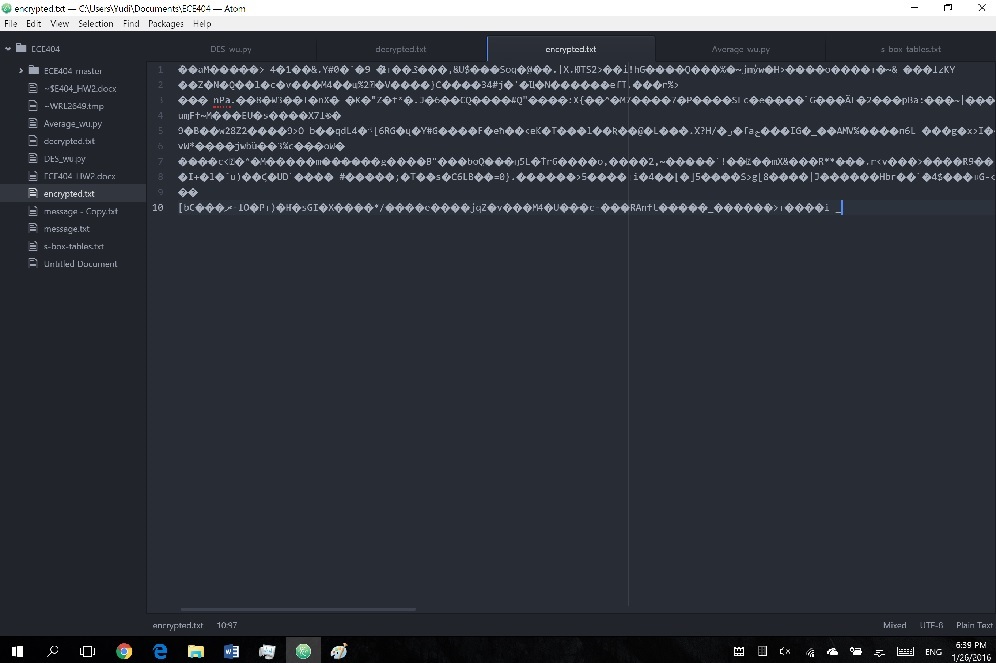
message.txt

On December 18, 2013, security expert Brian Krebs broke news that Target was investigating a major data breach "potentially involving millions of customer credit and debit card records." The report quickly spread across news channels. On December 19, Target confirmed the incident via a press release, revealing that the hack took place between November 27 and December 15, 2013. Target warned that up to 40 million consumer credit and debit cards may have been compromised. Hackers gained access to customer names, card numbers, expiration dates, and CVV security codes of the cards issued by financial institutions. On December 27 Target disclosed that debit card PIN data had also been stolen, albeit in encrypted form, reversing an earlier stance that PIN data was not part of the breach. Target noted that the accessed PIN numbers were encrypted using Triple DES and has stated the PINs remain "safe and secure" due to the encryption. On January 10, 2014, Target disclosed that the names, mailing addresses, phone numbers or email addresses of up to 70 million additional people had also been stolen, bringing the possible number of customers affected up to 110 million.

Key

ecepurdu

encrypted.txt



decrypted.txt

On December 18, 2013, security expert Brian Krebs broke news that Target was investigating a major data breach "potentially involving millions of customer credit and debit card records." The report quickly spread across news channels. On December 19, Target confirmed the incident via a press release, revealing that the hack took place between November 27 and December 15, 2013. Target warned that up to 40 million consumer credit and debit cards may have been compromised. Hackers gained access to customer names, card numbers, expiration dates, and CVV security codes of the cards issued by financial institutions. On December 27 Target disclosed that debit card PIN data had also been stolen, albeit in encrypted form, reversing an earlier stance that PIN data was not part of the breach. Target noted that the accessed PIN numbers were encrypted using Triple DES and has stated the PINs remain "safe and secure" due to the encryption. On January 10, 2014, Target disclosed that the names, mailing addresses, phone numbers or email addresses of up to 70 million additional people had also been stolen, bringing the possible number of customers affected up to 110 million.

Problem 2

Code

#!/usr/bin/env/python

### Average\_wu.py

import sys

import random

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:

arrays = []

with open('s-box-tables.txt') as f:

for line in f:

if len(line) > 4:

row = line.split()

arrays.append(row)

s\_box = []

for i in range(0,32, 4):

s\_box.append([arrays[k] for k in range(i, i+4)]) # S\_BOX

f.close()

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

def get\_encryption\_key(): # key

## ask user for input

while True:

key = raw\_input("Please enter a 8 characters key: ")

## make sure it satisfies any constraints on the key

if len(key) == 8:

break

else:

print("Your key is not 8 characters long")

user\_key\_bv = BitVector(textstring = key)

key\_bv = user\_key\_bv.permute(key\_permutation\_1) ## permute() is a BitVector function

return key\_bv

################################# Generatubg round keys ########################

def extract\_round\_key( nkey ): # round key

round\_key = []

for i in range(16):

[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

nkey.permute(key\_permutation\_2)

round\_key.append(nkey)

return round\_key

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

def des(LE,RE,round\_key):

for i in range(16):

##E-step

temp3 = RE

temp = RE.permute(expansion\_permutation)

temp = temp ^ round\_key[i]

##S-BOX

temp2 = BitVector(size = 0)

for i in range(8):

row = 2 \* temp[i \* 6] + temp[i \* 6 + 5]

col = 8 \* temp[i \* 6 + 1] + 4 \* temp[i \* 6 + 2] + 2 \* temp[i \* 6 + 3] + temp[i \* 6 + 4]

temp2 += BitVector(intVal = int(s\_box[i][row][col]), size = 4)

##P-BOX

temp2 = temp2.permute(p\_box\_permutation)

RE = LE ^ temp2

LE = temp3

##output

combined = RE + LE

return combined

#################################### main #######################################

def main():

## prompts the user for the key

key = get\_encryption\_key()

input\_file = 'message.txt'

##problem II part I

bv = BitVector( filename = input\_file )

block\_ct = 0

total\_diff = 0

round\_key = extract\_round\_key(key)

while(bv.more\_to\_read):

bitvec = bv.read\_bits\_from\_file( 64 )

##count how many plaintext blocks

block\_ct += 1

##make sure bitvec is 64 bits, if not, pad from right

bit\_ct = bitvec.length()

if(bit\_ct != 64):

bitvec.pad\_from\_right(64 - bit\_ct)

##modify one random bit

change\_bit = BitVector(intVal = (1 << random.randint(0,63)),size = 64)

bitvec2 = bitvec ^ change\_bit

[LE, RE] = bitvec.divide\_into\_two()

[LE2, RE2] = bitvec2.divide\_into\_two()

cypher1 = des(LE,RE,round\_key)

cypher2 = des(LE2,RE2,round\_key)

diff\_bits = (cypher1 ^ cypher2).count\_bits()

total\_diff += diff\_bits

##find average difference on one block

avg\_diff = total\_diff / block\_ct

print "Average ciphertext change for plaintext change is %d" % avg\_diff

##problem II part II

##store original cipher in result1

bv = BitVector( filename = input\_file )

result1 = BitVector(size = 0)

while(bv.more\_to\_read):

bitvec = bv.read\_bits\_from\_file( 64 )

bit\_ct = bitvec.length()

if(bit\_ct != 64):

bitvec.pad\_from\_right(64 - bit\_ct)

[LE, RE] = bitvec.divide\_into\_two()

cypher1 = des(LE,RE,round\_key)

result1 += cypher1

##randomly change S\_box for the first time and store result in result2

for i in range(8):

for j in range(4):

row = []

row = random.sample(range(16),16)

s\_box[i][j] = row

bv = BitVector( filename = input\_file )

result2 = BitVector(size = 0)

while(bv.more\_to\_read):

bitvec = bv.read\_bits\_from\_file( 64 )

bit\_ct = bitvec.length()

if(bit\_ct != 64):

bitvec.pad\_from\_right(64 - bit\_ct)

[LE, RE] = bitvec.divide\_into\_two()

cypher2 = des(LE,RE,round\_key)

result2 += cypher2

##randomly change S\_box for the second time and store result in result3

for i in range(8):

for j in range(4):

row = []

row = random.sample(range(16),16)

s\_box[i][j] = row

bv = BitVector( filename = input\_file )

result3 = BitVector(size = 0)

while(bv.more\_to\_read):

bitvec = bv.read\_bits\_from\_file( 64 )

bit\_ct = bitvec.length()

if(bit\_ct != 64):

bitvec.pad\_from\_right(64 - bit\_ct)

[LE, RE] = bitvec.divide\_into\_two()

cypher3 = des(LE,RE,round\_key)

result3 += cypher3

##find differences and find average for one block

diff\_bits1 = (result1 ^ result2).count\_bits()

diff\_bits2 = (result1 ^ result3).count\_bits()

avg\_diff = (diff\_bits1 + diff\_bits2) / 2 / block\_ct

print "Average ciphertext change for S\_box change is %d" % avg\_diff

##problem II part IIII

total\_key = 0

##try 5 different modified keys

for ct in range(5):

bv = BitVector( filename = input\_file )

change\_bit = BitVector(intVal = (1 << random.randint(0,63)),size = 64)

alter\_key = key ^ change\_bit

round\_key = extract\_round\_key(alter\_key)

result4 = BitVector(size = 0)

while(bv.more\_to\_read):

bitvec = bv.read\_bits\_from\_file( 64 )

bit\_ct = bitvec.length()

if(bit\_ct != 64):

bitvec.pad\_from\_right(64 - bit\_ct)

[LE, RE] = bitvec.divide\_into\_two()

cypher4 = des(LE,RE,round\_key)

result4 += cypher4

diff\_key = (result1 ^ result4).count\_bits()

total\_key += diff\_key

avg\_key = total\_key / 5 / block\_ct

print "Average ciphertext change for key change is %d" % avg\_key

if \_\_name\_\_ == "\_\_main\_\_":

main()

Output

For effects of diffusion on changing one bit in plaintext, I change one bit for every plaintext block, and add all difference together then divided by number of blocks. Result is 23 bits.

For effects of diffusion on changing s\_box, I put random integers between 0-16 on every row, and run encryption, compare difference with the original file, and twice. Then, compute the average change on every plaintext block. Result is 32 bits.

For effects of confusion on changing one bit in the key, I change one bit and compare the result with the original output, for five times, then compute the average change on every plaintext block. Result is 32bits.

But result changes on different runs, because it’s not depending on large number of runs.

yudi3160@ubuntu:~/Documents/ECE404$ python Average\_wu.py

Please enter a 8 characters key: ecepurdu

Average ciphertext change for plaintext change is 23

Average ciphertext change for S\_box change is 32

Average ciphertext change for key change is 32