NAGALAND UNIVERSITY

SCHOOL OF ENGINEERING & TECHNOLOGY

LAB MANUAL

CRYPTOGRAPHY & INFORMATION SECURITY LAB

SHANCHAMO YANTHAN

ASSISTANT PROFESSOR

DEPARTMENT OF INFORMATION TECHNOLOGY

TABLE OF CONTENTS

*Write all the programs in python

LAB EXPERIMENT 1

1. Implement the monoalphabetic substitution encryption and decryption techniques. The program must work with any key.

```
def subcipher(string, shift):
  string=string.lower()
  cipher = ''
  for char in string:
    if char == ' ':
      cipher = cipher + char
    else:
      cipher = cipher + chr((ord(char) + shift - 97) % 26 + 97)
  return cipher
text = input("enter the plaintext: ")
s = int(input("enter shift number: "))
print("original plaintext: ", text)
cipher=encrypt(text, s)
print("after encryption: ", subcipher.upper())
```

```
enter the plaintext: hello
enter shift number: 3
original plaintext:  hello
after encryption:  KHOOR
```

Figure 1(a) – Substitution Encryption.

```
def subdecrypt(string, shift):
  string=string.upper()
  cipher = ''
  for char in string:
    if char == ' ':
      cipher = cipher + char
    else:
      cipher = cipher + chr((ord(char) - shift - 65) % 26 + 65)
  return cipher
cipher = input("enter the ciphertext: ")
s = int(input("enter shift number: "))
print("original cipher: ", cipher)
print("after decryption: ", subdecrypt(cipher, s).lower())
```

```
enter the ciphertext: khoor
enter shift number: 3
original cipher:  khoor
after decryption:  hello
```

Figure 1(b) - Substitution Decryption.

LAB EXPERIMENT 2

2. Implement the Vignere Polyalphabetic substitution encryption and decryption techniques. The user must have option to enter the plaintext and key of his or her choice.

```
def polyencrypt(string, key):
  string=string.lower()
  cipher = ''
  klen=len(key)
  count=0
  for char in string:
    if char == ' ':
      cipher = cipher + char
    else:
      cipher = cipher + chr((ord(char) - 97 + ord(key[count])-97) % 26 + 97)
      count+=1
    if count==klen:
        count=0
  return cipher.upper()
text = input("enter the plaintext: ")
key = input("enter key: ")
print("original plaintext: ", text)
print("after encryption: ", polyencrypt(text, key))
```

```
enter the plaintext: hello
enter key: hello
original plaintext:  hello
after encryption:  OIWWC
```

Figure 2(a) – Polyalphabetic Encryption

```
def polydecrypt(string, key):
  string=string.upper()
  key=key.upper()
  cipher = ''
  klen=len(key)
  count=0
  for char in string:
    if char == ' ':
      cipher = cipher + char
    else:
      cipher = cipher + chr((ord(char) - 65 - ord(key[count])-65) % 26 + 65)
      count+=1
    if count==klen:
        count=0
  return cipher.lower()
ctext = input("enter the ciphertext: ")
key = input("enter key: ")
print("original ciphertext: ", ctext)
print("after decryption: ", polydecrypt(ctext, key))
```

```
enter the ciphertext: OIWWC
enter key: hello
original ciphertext:  OIWWC
after decryption:  hello
```

Figure 2(b) – Polyalphabetic Decryption

LAB EXPERIMENT 3

3. Modify the program no. 2 and implement the autokey polyalphabetic encryption and decryption technique.

```
def polyencrypt(string, key):
  string=string.lower()
  cipher = ''
  klen=len(key)
  count=0
  for char in string:
    if char == ' ':
      cipher = cipher + char
    else:
      if count<klen:
        k=key[count]
      else:
        k=string[count-klen]
      cipher = cipher + chr((ord(char) - 97 + ord(k)-97) % 26 + 97)
      count+=1
  return cipher.upper()
text = input("enter the plaintext: ")
key = input("enter key: ")
print("original plaintext: ", text)
print("after encryption: ", polyencrypt(text, key))
```

```
enter the plaintext: wearediscoveredsaveyourself
enter key: deceptive
original plaintext:  wearediscoveredsaveyourself
after encryption:  ZICVTWQNGKZEIIGASXSTSLVVWLA
```

Figure 3(a) – Autokey Encryption

```
def polydecrypt(string, key):
  string=string.upper()
  key=key.upper()
  plain = ''
  klen=len(key)
  count=0
  for char in string:
    if char == ' ':
      plain = plain + char
    else:
      if count<klen:
        k=key[count]
      else:
        k=plain[count-klen]
      plain = plain + chr((ord(char) - 65 - ord(k)-65) % 26 + 65)
      count+=1
  return plain.lower()
text = input("enter the Ciphertext: ")
key = input("enter key: ")
print("original Ciphertext: ", text)
print("after Decryption: ", polydecrypt(text, key))
```

```
enter the Ciphertext: ZICVTWQNGKZEIIGASXSTSLVVWLA
enter key: deceptive
original Ciphertext:  ZICVTWQNGKZEIIGASXSTSLVVWLA
after Decryption:  wearediscoveredsaveyourself
```

Figure 3(b) – Autokey Decryption

LAB EXPERIMENT 4

4. Implement the playfair encryption and decryption techniques.

```
key=input("Enter key")
key=key.replace(" ", "")
key=key.upper()
def matrix(x,y,initial):
    return [[initial for i in range(x)] for j in range(y)]

result=list()
for c in key: #storing key
    if c not in result:
        if c=='J':
            result.append('I')
        else:
            result.append(c)
flag=0
for i in range(65,91): #storing other character
    if chr(i) not in result:
        if i==73 and chr(74) not in result:
            result.append("I")
            flag=1
        elif flag==0 and i==73 or i==74:
            pass
        else:
            result.append(chr(i))
k=0
my_matrix=matrix(5,5,0) #initialize matrix
for i in range(0,5): #making matrix
    for j in range(0,5):
        my_matrix[i][j]=result[k]
        k+=1
```

Figure 4(a) – Play fair Encryption

```python
def locindex(c): #get location of each character
    loc=list()
    if c=='J':
        c='I'
    for i ,j in enumerate(my_matrix):
        for k,l in enumerate(j):
            if c==l:
                loc.append(i)
                loc.append(k)
                return loc

def encrypt():  #Encryption
    msg=str(input("ENTER MSG:"))
    msg=msg.upper()
    msg=msg.replace(" ", "")
    i=0
    for s in range(0,len(msg)+1,2):
        if s<len(msg)-1:
            if msg[s]==msg[s+1]:
                msg=msg[:s+1]+'X'+msg[s+1:]
    if len(msg)%2!=0:
        msg=msg[:]+'X'
    print("CIPHER TEXT:",end=' ')
    while i<len(msg):
        loc=list()
        loc=locindex(msg[i])
        loc1=list()
        loc1=locindex(msg[i+1])
        if loc[1]==loc1[1]:
            print("{}{}".format(my_matrix[(loc[0]+1)%5][loc[1]],my_matrix[(loc1[0]+1)%5][loc1[1]]),end=' ')
        elif loc[0]==loc1[0]:
            print("{}{}".format(my_matrix[loc[0]][(loc[1]+1)%5],my_matrix[loc1[0]][(loc1[1]+1)%5]),end=' ')
        else:
            print("{}{}".format(my_matrix[loc[0]][loc1[1]],my_matrix[loc1[0]][loc[1]]),end=' ')
        i=i+2
```

Figure 4(b) – continued

```python
def decrypt():  #decryption
    msg=str(input("ENTER CIPHER TEXT:"))
    msg=msg.upper()
    msg=msg.replace(" ", "")
    print("PLAIN TEXT:",end=' ')
    i=0
    while i<len(msg):
        loc=list()
        loc=locindex(msg[i])
        loc1=list()
        loc1=locindex(msg[i+1])
        if loc[1]==loc1[1]:
            print("{}{}".format(my_matrix[(loc[0]-1)%5][loc[1]],my_matrix[(loc1[0]-1)%5][loc1[1]]),end=' ')
        elif loc[0]==loc1[0]:
            print("{}{}".format(my_matrix[loc[0]][(loc[1]-1)%5],my_matrix[loc1[0]][(loc1[1]-1)%5]),end=' ')
        else:
            print("{}{}".format(my_matrix[loc[0]][loc1[1]],my_matrix[loc1[0]][loc[1]]),end=' ')
        i=i+2

while(1):
    choice=int(input("\n 1.Encryption \n 2.Decryption: \n 3.EXIT"))
    if choice==1:
        encrypt()
    elif choice==2:
        decrypt()
    elif choice==3:
        exit()
    else:
        print("Choose correct choice")
```

Figure 4(c) – continued

LAB EXPERIMENT 5

5. Implement the Hill cipher. Show the encryption of the message "attack is tonight" using the matrix.

$$\begin{vmatrix} 3 & 10 & 20 \\ 20 & 9 & 17 \\ 9 & 4 & 17 \end{vmatrix}$$

```python
def hill_encrypt(ptext):
    ptext=ptext.lower()
    print("Plaintext is : ",ptext)
    key_matrix=[[17,17,5], [21,18,21], [2,2,19]]
    print("Key matrix: ")
    for row in key_matrix:
        for val in row:
            print (val,end=' ')
        print ("\n")
    i=0
    cipher = ''
    for i in range(0, len(ptext), 3):
        p=ptext[i:i+3]
        row=0
        col=0
        for col in range(3):
            c=0
            for row in range(3):
                #print(p," ",row," ",col)
                c+=(ord(p[row])-97)*key_matrix[row][col]
                #print(p[row]," ",(ord(p[row])-97)," ",key_matrix[row][col])
            cipher=cipher+chr(c%26+97)
            #print(cipher)
    return cipher.upper()
print ("Hill Cipher ")
plaintext=input("Enter the plaintext: ")
print("Cipher text: ",hill_encrypt(plaintext))
```

```
Hill Cipher
Enter the plaintext: paymoremoney
Plaintext is :  paymoremoney
Key matrix:
17 17 5

21 18 21

2 2 19

Cipher text:  RRLMWBKASPDH
```

Figure 5(a) – Hill Cipher Encryption

```
def hill_decrypt(ctext):
    ctext=ctext.upper()
    print("Plaintext is : ",ctext)
    key_matrix=[[4,9,15], [15,17,6], [24,0,17]]
    print("Key matrix: ")
    for row in key_matrix:
        for val in row:
            print (val,end=' ')
        print ("\n")
    i=0
    plaintext = ''
    for i in range(0, len(ctext), 3):
        c=ctext[i:i+3]
        row=0
        col=0
        for col in range(3):
            p=0
            for row in range(3):
                p+=(ord(c[row])-65)*key_matrix[row][col]
            plaintext=plaintext+chr(p%26+65)
    return plaintext.lower()
print ("Hill Cipher ")
ciphertext=input("Enter the ciphertext: ")
print("plaintext: ",hill_decrypt(ciphertext))
```

```
Hill Cipher
Enter the ciphertext: RRLMWBKASPDH
Plaintext is :  RRLMWBKASPDH
Key matrix:
4 9 15

15 17 6

24 0 17

plaintext:  paymoremoney
```

Figure 5(b) – Hill Cipher Decryption

LAB EXPERIMENT 6

6. Implement the DES algorithm.

```
#Initial permut matrix for the datas
PI = [58, 50, 42, 34, 26, 18, 10, 2,
      60, 52, 44, 36, 28, 20, 12, 4,
      62, 54, 46, 38, 30, 22, 14, 6,
      64, 56, 48, 40, 32, 24, 16, 8,
      57, 49, 41, 33, 25, 17, 9, 1,
      59, 51, 43, 35, 27, 19, 11, 3,
      61, 53, 45, 37, 29, 21, 13, 5,
      63, 55, 47, 39, 31, 23, 15, 7]

#Initial permut made on the key
CP_1 = [57, 49, 41, 33, 25, 17, 9,
        1, 58, 50, 42, 34, 26, 18,
        10, 2, 59, 51, 43, 35, 27,
        19, 11, 3, 60, 52, 44, 36,
        63, 55, 47, 39, 31, 23, 15,
        7, 62, 54, 46, 38, 30, 22,
        14, 6, 61, 53, 45, 37, 29,
        21, 13, 5, 28, 20, 12, 4]

#Permut applied on shifted key to get Ki+1
CP_2 = [14, 17, 11, 24, 1, 5, 3, 28,
        15, 6, 21, 10, 23, 19, 12, 4,
        26, 8, 16, 7, 27, 20, 13, 2,
        41, 52, 31, 37, 47, 55, 30, 40,
        51, 45, 33, 48, 44, 49, 39, 56,
        34, 53, 46, 42, 50, 36, 29, 32]

#Expand matrix to get a 48bits matrix of datas to apply the xor with Ki
E = [32, 1, 2, 3, 4, 5,
     4, 5, 6, 7, 8, 9,
     8, 9, 10, 11, 12, 13,
     12, 13, 14, 15, 16, 17,
     16, 17, 18, 19, 20, 21,
     20, 21, 22, 23, 24, 25,
     24, 25, 26, 27, 28, 29,
     28, 29, 30, 31, 32, 1]
```

Figure 6(a) – DES Encryption

```
#SBOX
S_BOX = [
[[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],],
[[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],],
[[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],],
[[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],],
[[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],],
[[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],],
[[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],],
[[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],]]
```

Figure 6(b) – DES Encryption..continued

```
#Permutation made after each SBox substitution for each round
P = [16, 7, 20, 21, 29, 12, 28, 17,
     1, 15, 23, 26, 5, 18, 31, 10,
     2, 8, 24, 14, 32, 27, 3, 9,
     19, 13, 30, 6, 22, 11, 4, 25]

#Final permut for datas after the 16 rounds
PI_1 = [40, 8, 48, 16, 56, 24, 64, 32,
        39, 7, 47, 15, 55, 23, 63, 31,
        38, 6, 46, 14, 54, 22, 62, 30,
        37, 5, 45, 13, 53, 21, 61, 29,
        36, 4, 44, 12, 52, 20, 60, 28,
        35, 3, 43, 11, 51, 19, 59, 27,
        34, 2, 42, 10, 50, 18, 58, 26,
        33, 1, 41, 9, 49, 17, 57, 25]

#Matrix that determine the shift for each round of keys
SHIFT = [1,1,2,2,2,2,2,2,1,2,2,2,2,2,2,1]

def string_to_bit_array(text):#Convert a string into a list of bits
    array = list()
    for char in text:
        binval = binvalue(char, 8)#Get the char value on one byte
        array.extend([int(x) for x in list(binval)]) #Add the bits to the final list
    return array

def bit_array_to_string(array): #Recreate the string from the bit array
    res = ''.join([chr(int(y,2)) for y in [''.join([str(x) for x in _bytes]) for _bytes in  nsplit(array,8)]])
    return res
```

Figure 6(c) – DES Encryption..continued

```
def binvalue(val, bitsize): #Return the binary value as a string of the given size
    binval = bin(val)[2:] if isinstance(val, int) else bin(ord(val))[2:]
    if len(binval) > bitsize:
        raise "binary value larger than the expected size"
    while len(binval) < bitsize:
        binval = "0"+binval #Add as many 0 as needed to get the wanted size
    return binval
def nsplit(s, n):#Split a list into sublists of size "n"
    return [s[k:k+n] for k in range(0, len(s), n)]
ENCRYPT=1
DECRYPT=0
class des():
    def __init__(self):
        self.password = None
        self.text = None
        self.keys = list()

    def run(self, key, text, action=ENCRYPT, padding=False):
        if len(key) < 8:
            raise "Key Should be 8 bytes long"
        elif len(key) > 8:
            key = key[:8] #If key size is above 8bytes, cut to be 8bytes long

        self.password = key
        self.text = text

        if padding and action==ENCRYPT:
            self.addPadding()
        elif len(self.text) % 8 != 0:#If not padding specified data size must be multiple of 8 bytes
            raise "Data size should be multiple of 8"
        self.generatekeys() #Generate all the keys
        text_blocks = nsplit(self.text, 8) #Split the text in blocks of 8 bytes so 64 bits
        result = list()
```

Figure 6(d) – DES Encryption..continued

```
    for block in text_blocks:#Loop over all the blocks of data
        block = string_to_bit_array(block)#Convert the block in bit array
        block = self.permut(block,PI)#Apply the initial permutation
        g, d = nsplit(block, 32) #g(LEFT), d(RIGHT)
        tmp = None
        for i in range(16): #Do the 16 rounds
            d_e = self.expand(d, E) #Expand d to match Ki size (48bits)
            if action == ENCRYPT:
                tmp = self.xor(self.keys[i], d_e)#If encrypt use Ki
            else:
                tmp = self.xor(self.keys[15-i], d_e)#If decrypt start by the last key
            tmp = self.substitute(tmp) #Method that will apply the SBOXes
            tmp = self.permut(tmp, P)
            tmp = self.xor(g, tmp)
            g = d
            d = tmp
        result += self.permut(d+g, PI_1) #Do the last permut and append the result to result
    final_res = bit_array_to_string(result)
    if padding and action==DECRYPT:
        return self.removePadding(final_res) #Remove the padding if decrypt and padding is true
    else:
        return final_res #Return the final string of data ciphered/deciphered

def substitute(self, d_e):#Substitute bytes using SBOX
    subblocks = nsplit(d_e, 6)#Split bit array into sublist of 6 bits
    result = list()
    for i in range(len(subblocks)): #For all the sublists
        block = subblocks[i]
        row = int(str(block[0])+str(block[5]),2)#Get the row with the first and last bit
        column = int(''.join([str(x) for x in block[1:][:-1]]),2) #Column is the 2,3,4,5th bits
        val = S_BOX[i][row][column] #Take the value in the SBOX appropriated for the round (i)
        bin = binvalue(val, 4)#Convert the value to binary
        result += [int(x) for x in bin]#And append it to the resulting list
    return result
```

Figure 6(e) – DES Encryption..continued

```
    def permut(self, block, table):#Permut the given block using the given table (so generic method)
        return [block[x-1] for x in table]
    def expand(self, block, table):#Do the exact same thing than permut but for more clarity has been renamed
        return [block[x-1] for x in table]
    def xor(self, t1, t2):#Apply a xor and return the resulting list
        return [x^y for x,y in zip(t1,t2)]
    def generatekeys(self):#Algorithm that generates all the keys
        self.keys = []
        key = string_to_bit_array(self.password)
        key = self.permut(key, CP_1) #Apply the initial permut on the key
        g, d = nsplit(key, 28) #Split it in to (g->LEFT),(d->RIGHT)
        for i in range(16):#Apply the 16 rounds
            g, d = self.shift(g, d, SHIFT[i]) #Apply the shift associated with the round (not always 1)
            tmp = g + d #Merge them
            self.keys.append(self.permut(tmp, CP_2)) #Apply the permut to get the Ki
    def shift(self, g, d, n): #Shift a list of the given value
        return g[n:] + g[:n], d[n:] + d[:n]
    def addPadding(self):#Add padding to the datas using PKCS5 spec.
        pad_len = 8 - (len(self.text) % 8)
        self.text += pad_len * chr(pad_len)
    def removePadding(self, data):#Remove the padding of the plain text (it assume there is padding)
        pad_len = ord(data[-1])
        return data[:-pad_len]
    def encrypt(self, key, text, padding=False):
        return self.run(key, text, ENCRYPT, padding)
    def decrypt(self, key, text, padding=False):
        return self.run(key, text, DECRYPT, padding)
if __name__ == '__main__':
    key = "secret_k"
    text= "Hello wo"
    d = des()
    r = d.encrypt(key,text)
    r2 = d.decrypt(key,r)
    print("Ciphered: %r" % r)
    print("Deciphered: ", r2)
```

Figure 6(f) – DES Encryption..continued