#Caesar Cipher:

```
lower='abcdefghijklmnopqrstuvwxy'
upper='ABCDEFGHIJKLMNOPQRSTUVWXYZ'
encrypt,decrypt="",""
print("Enter message")
message=input()
print("enter shift:")
k=int(input())
for i in message:
    if i in lower:
        newpos=(lower.find(i)+k)%len(lower)
        encrypt+=lower[newpos]
for i in message:
    if i in upper:
        newpos=(lower.find(i)+k)%len(lower)
        encrypt+=lower[newpos]
for i in encrypt:
    if i in lower:
        newpos=(lower.find(i)-k)%len(lower)
        decrypt+=lower[newpos]
for i in encrypt:
    if i in upper:
        newpos=(lower.find(i)-k)%len(lower)
        decrypt+=lower[newpos]
print("Encrypted message:",encrypt)
print("Decrypted message:”,decrypt)
```

#Playfair Cipher

```
# playfair cipher

def create_matrix(key):
    key=key.upper()
    matrix=[[0 for i in range(5)] for j in range(5)]
    letter_added=[]
    row=0
    col=0

    for letter in key:
        if letter not in letter_added:
            matrix[row][col]=letter
            letter_added.append(letter)
        else:
            continue
        if(col==4):
            col=0
            row+=1
        else:
```

```
            col+=1
    for letter in range(65,91):
        if(letter == 74):
            continue
        if chr(letter) not in letter_added:
            letter_added.append(chr(letter))
    index=0
    for i in range(5):
        for j in range(5):
            matrix[i][j] = letter_added[index]
            index+=1
    return matrix


#Add fillers if the same letter is in a pair
def separate_same_letters(message):
    index = 0
    while (index<len(message)):
        l1 = message[index]
        if index == len(message)-1:
            message = message + 'X'
            index += 2
            continue
        l2 = message[index+1]
        if l1==l2:
            message = message[:index+1] + "X" + message[index+1:]
        index +=2
    return message

#Return the index of a letter in the matrix
#This will be used to know what rule (1-4) to apply
def indexOf(letter,matrix):
    for i in range (5):
        try:
            index = matrix[i].index(letter)
            return (i,index)
        except:
            continue
#Implementation of the playfair cipher
#If encrypt=True the method will encrypt the message
# otherwise the method will decrypt
def playfair(key, message, encrypt=True):
    inc = 1
    if encrypt==False:
        inc = -1
    matrix = create_matrix(key)
    message = message.upper()
    message = message.replace(' ','')
    message = separate_same_letters(message)
    cipher_text=''
    for (l1, l2) in zip(message[0::2], message[1::2]):
        row1,col1 = indexOf(l1,matrix)
```

```
        row2,col2 = indexOf(l2,matrix)
        if row1==row2: #Rule 2, the letters are in the same row
            cipher_text += matrix[row1][(col1+inc)%5] +
matrix[row2][(col2+inc)%5]
        elif col1==col2:# Rule 3, the letters are in the same
column
            cipher_text += matrix[(row1+inc)%5][col1] +
matrix[(row2+inc)%5][col2]
        else: #Rule 4, the letters are in a different row and
column
            cipher_text += matrix[row1][col2] + matrix[row2][col1]

    return cipher_text
if __name__=='__main__':
    # a sample of encryption and decryption
    print ('Encripting')
    print ( playfair('secret', 'my secret message'))
    print ('Decrypting')
    print ( playfair('secret', 'LZECRTCSITCVAHBT', False))



#vigenere cipher


def generateKey(string, key):
  key = list(key)
  if len(string) == len(key):
    return(key)
  else:
    for i in range(len(string) -len(key)):
      key.append(key[i % len(key)])
  return("" . join(key))

def encryption(string, key):
  encrypt_text = []
  for i in range(len(string)):
    x = (ord(string[i]) +ord(key[i])) % 26
    x += ord('A')
    encrypt_text.append(chr(x))
  return("" . join(encrypt_text))

def decryption(encrypt_text, key):
  orig_text = []
```

```
  for i in range(len(encrypt_text)):
    x = (ord(encrypt_text[i]) -ord(key[i]) + 26) %
26
    x += ord('A')
    orig_text.append(chr(x))
  return("" . join(orig_text))

if __name__ == "__main__":
  string = input("Enter the message: ")
  keyword = input("Enter the keyword: ")
  key = generateKey(string, keyword)
  encrypt_text = encryption(string,key)
  print("Encrypted message:", encrypt_text)
  print("Decrypted message:",
decryption(encrypt_text, key))
```

WEEK 2

#Transposition cipher

```
import math
key=input("Enter keyword text (Contains unique letters only):
").lower().replace(" ", "")
plain_text = input("Enter plain text (Letters only):
").lower().replace(" ", "")

len_key = len(key)
len_plain = len(plain_text)
row = int(math.ceil(len_plain / len_key))
matrix = [ ['X']*len_key for i in range(row) ]

# print(matrix)

#ENCRYPTION
t = 0
for r in range(row):
  for c,ch in enumerate(plain_text[t : t+ len_key]):
    matrix[r][c] = ch
  t += len_key

# print(matrix)
sort_order = sorted([(ch,i) for i,ch in enumerate(key)])  #to make
alphabetically order of chars
```

```
# print(sort_order)

cipher_text = ''
for ch,c in sort_order:
  for r in range(row):
    cipher_text += matrix[r][c]

print("Encryption")
print("Plain text is :",plain_text)
print("Cipher text is:",cipher_text)

#DECRYPTION

matrix_new = [ ['X']*len_key for i in range(row) ]
key_order = [ key.index(ch) for ch in sorted(list(key))]  #to make
original key order when we know keyword
# print(key_order)

t = 0
for c in key_order:
  for r,ch in enumerate(cipher_text[t : t+ row]):
    matrix_new[r][c] = ch
  t += row
# print(matrix_new)

p_text = ''
for r in range(row):
  for c in range(len_key):
    p_text += matrix_new[r][c] if matrix_new[r][c] != 'X' else ''

print("Decryption")
print("Cipher text is:",cipher_text)
print("Plain text is :",p_text)


#Affine cipher


def modMulInv(a,n):
    inv = 0
    for i in range(1,n):
        if(((a%n)*(i%n))%n == 1):
            inv = i
            break
    return inv
plainText = input("Enter Plain Text: ").upper()
a = int(input("Enter first key(a): "))
b = int(input("Enter second key(b): "))
```

```
cryptText = ""
for letter in plainText:
    if letter == ' ':
        cryptText = cryptText + " "
    else:
        cryptText = cryptText +
chr((((ord(letter)-65)*a+b)%26)+65)
print("Encrypted Text: " + cryptText)
aInv = modMulInv(a,26)
if(aInv==0):
    print("No multiplicative inverse for a")
    exit(0)
cryptText1 = input("Enter Encrypted Text:
").upper()
plainText1 = ""
for letter in cryptText1:
    if letter == ' ':
        plainText1 = plainText1 + " "
    else:
        plainText1 = plainText1 +
chr((((ord(letter)-65-b)*aInv)%26)+65)
print("Plain Text: "+plainText1)


#bruteforce affine cipher


def modMulInv(a,n):
    inv = 0
    for i in range(1,n):
        if(((a%n)*(i%n))%n == 1):
            inv = i
            break
    return inv

plainText = input("Enter Plain Text: ").upper()
```

```
cryptText = input("Enter Encrypted Text:
").upper()

aPossible = [1,3,5,7,9,11,15,17,19,21,23,25]

for a in aPossible:
    for b in range(1,26):
        temp = ""
        aInv = modMulInv(a,26)

        for letter in cryptText:
            if letter == ' ':
                temp = temp + " "
            else:
                temp = temp +
chr((((ord(letter)-65-b)*aInv)%26)+65)
        if temp == plainText:
            print("Keys found, they are: "+str(a)
+" and "+str(b))
            exit(0)
print("KEYS NOT FOUND")

#DES


def hex2bin(s):
    mp = {'0' : "0000",
        '1' : "0001",
        '2' : "0010",
        '3' : "0011",
        '4' : "0100",
        '5' : "0101",
        '6' : "0110",
        '7' : "0111",
        '8' : "1000",
        '9' : "1001",
        'A' : "1010",
        'B' : "1011",
```

```
        'C' : "1100",
        'D' : "1101",
        'E' : "1110",
        'F' : "1111" }
    bin = ""
    for i in range(len(s)):
        bin = bin + mp[s[i]]
    return bin
def bin2hex(s):
    mp = {"0000" : '0',
        "0001" : '1',
        "0010" : '2',
        "0011" : '3',
        "0100" : '4',
        "0101" : '5',
        "0110" : '6',
        "0111" : '7',
        "1000" : '8',
        "1001" : '9',
        "1010" : 'A',
        "1011" : 'B',
        "1100" : 'C',
        "1101" : 'D',
        "1110" : 'E',
        "1111" : 'F' }
    hex = ""
    for i in range(0,len(s),4):
        ch = ""
        ch = ch + s[i]
        ch = ch + s[i + 1]
        ch = ch + s[i + 2]
        ch = ch + s[i + 3]
        hex = hex + mp[ch]

    return hex

def bin2dec(binary):
    decimal, i= 0, 0
```

```
    while(binary != 0):
        dec = binary % 10
        decimal = decimal + dec * pow(2, i)
        binary = binary//10
        i += 1
    return decimal

def dec2bin(num):
    res = bin(num).replace("0b", "")
    if(len(res)%4 != 0):
        div = len(res) / 4
        div = int(div)
        counter =(4 * (div + 1)) - len(res)
        for i in range(0, counter):
            res = '0' + res
    return res


def permute(k, arr, n):
    permutation = ""
    for i in range(0, n):
        permutation = permutation + k[arr[i] - 1]
    return permutation


def shift_left(k, nth_shifts):
    s = ""
    for i in range(nth_shifts):
        for j in range(1,len(k)):
            s = s + k[j]
        s = s + k[0]
        k = s
        s = ""
    return k


def xor(a, b):
    ans = ""
```

```
    for i in range(len(a)):
        if a[i] == b[i]:
            ans = ans + "0"
        else:
            ans = ans + "1"
    return ans


initial_perm = [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]


exp_d = [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 ]

per = [ 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 ]


sbox = [[[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] ] ]


final_perm = [ 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 ]

def encrypt(pt, rkb, rk):
    pt = hex2bin(pt)


    pt = permute(pt, initial_perm, 64)
    print("After initial permutation",
bin2hex(pt))


    left = pt[0:32]
    right = pt[32:64]
    for i in range(0, 16):

        right_expanded = permute(right, exp_d, 48)


        xor_x = xor(right_expanded, rkb[i])


        sbox_str = ""
        for j in range(0, 8):
            row = bin2dec(int(xor_x[j * 6] +
xor_x[j * 6 + 5]))
            col = bin2dec(int(xor_x[j * 6 + 1] +
xor_x[j * 6 + 2] + xor_x[j * 6 + 3] + xor_x[j * 6
+ 4]))
            val = sbox[j][row][col]
            sbox_str = sbox_str + dec2bin(val)


        sbox_str = permute(sbox_str, per, 32)


        result = xor(left, sbox_str)
```

```
        left = result


        if(i != 15):
            left, right = right, left
        print("Round", i + 1, ":- ",
bin2hex(left), " ", bin2hex(right), " ", rk[i])


    combine = left + right


    cipher_text = permute(combine, final_perm, 64)
    return cipher_text

print("Data encryption standard (DES):-")

pt = input("Enter the plain text: ")
key = input("Enter the Key: ")


key = hex2bin(key)


keyp = [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 ]

key = permute(key, keyp, 56)

shift_table = [1, 1, 2, 2,
                2, 2, 2, 2,
                1, 2, 2, 2,
```

```
                2, 2, 2, 1 ]


key_comp = [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 ]

left = key[0:28]
right = key[28:56]

rkb = []
rk = []
for i in range(0, 16):

    left = shift_left(left, shift_table[i])
    right = shift_left(right, shift_table[i])


    combine_str = left + right


    round_key = permute(combine_str, key_comp, 48)

    rkb.append(round_key)
    rk.append(bin2hex(round_key))

print("Encryption: ")
cipher_text = bin2hex(encrypt(pt, rkb, rk))
print("Cipher Text : ",cipher_text)

print("Decryption: ")
rkb_rev = rkb[::-1]
rk_rev = rk[::-1]
```

```
text = bin2hex(encrypt(cipher_text, rkb_rev,
rk_rev))
print("Plain Text : ",text)
```

#rc4

```
def key_scheduling(key):
    sched = [i for i in range(0, 256)]

    i = 0
    for j in range(0, 256):
        i = (i + sched[j] + key[j % len(key)]) %
256

        tmp = sched[j]
        sched[j] = sched[i]
        sched[i] = tmp

    return sched


def stream_generation(sched):
    stream = []
    i = 0
    j = 0
    while True:
        i = (1 + i) % 256
        j = (sched[i] + j) % 256

        tmp = sched[j]
        sched[j] = sched[i]
```

```
        sched[i] = tmp

        yield sched[(sched[i] + sched[j]) % 256]


def encrypt(text, key):
    text = [ord(char) for char in text]
    key = [ord(char) for char in key]

    sched = key_scheduling(key)
    key_stream = stream_generation(sched)

    ciphertext = ''
    for char in text:
        enc = str(hex(char ^
next(key_stream))).upper()
        ciphertext += (enc)

    return ciphertext


def decrypt(ciphertext, key):
    ciphertext = ciphertext.split('0X')[1:]
    ciphertext = [int('0x' + c.lower(), 0) for c
in ciphertext]
    key = [ord(char) for char in key]

    sched = key_scheduling(key)
    key_stream = stream_generation(sched)

    plaintext = ''
    for char in ciphertext:
        dec = str(chr(char ^ next(key_stream)))
        plaintext += dec

    return plaintext
```

```
if __name__ == '__main__':
    ed = input('Enter E for Encrypt, or D for
Decrypt: ').upper()
    if ed == 'E':
        plaintext = input('Enter your plaintext:
')
        key = input('Enter your secret key: ')
        result = encrypt(plaintext, key)
        print('Result: ')
        print(result)
    elif ed == 'D':
        ciphertext = input('Enter your ciphertext:
')
        key = input('Enter your secret key: ')
        result = decrypt(ciphertext, key)
        print('Result: ')
        print(result)
    else:
        print('Error in input - try again.’)


#TRIPLE DES


def hex2bin(s):
    mp = {'0' : "0000",
        '1' : "0001",
        '2' : "0010",
        '3' : "0011",
        '4' : "0100",
        '5' : "0101",
        '6' : "0110",
        '7' : "0111",
        '8' : "1000",
        '9' : "1001",
        'A' : "1010",
        'B' : "1011",
        'C' : "1100",
```

```
        'D' : "1101",
        'E' : "1110",
        'F' : "1111" }
    bin = ""
    for i in range(len(s)):
        bin = bin + mp[s[i]]
    return bin
def bin2hex(s):
    mp = {"0000" : '0',
        "0001" : '1',
        "0010" : '2',
        "0011" : '3',
        "0100" : '4',
        "0101" : '5',
        "0110" : '6',
        "0111" : '7',
        "1000" : '8',
        "1001" : '9',
        "1010" : 'A',
        "1011" : 'B',
        "1100" : 'C',
        "1101" : 'D',
        "1110" : 'E',
        "1111" : 'F' }
    hex = ""
    for i in range(0,len(s),4):
        ch = ""
        ch = ch + s[i]
        ch = ch + s[i + 1]
        ch = ch + s[i + 2]
        ch = ch + s[i + 3]
        hex = hex + mp[ch]

    return hex

def bin2dec(binary):
    decimal, i= 0, 0
    while(binary != 0):
```

```
        dec = binary % 10
        decimal = decimal + dec * pow(2, i)
        binary = binary//10
        i += 1
    return decimal

def dec2bin(num):
    res = bin(num).replace("0b", "")
    if(len(res)%4 != 0):
        div = len(res) / 4
        div = int(div)
        counter =(4 * (div + 1)) - len(res)
        for i in range(0, counter):
            res = '0' + res
    return res


def permute(k, arr, n):
    permutation = ""
    for i in range(0, n):
        permutation = permutation + k[arr[i] - 1]
    return permutation


def shift_left(k, nth_shifts):
    s = ""
    for i in range(nth_shifts):
        for j in range(1,len(k)):
            s = s + k[j]
        s = s + k[0]
        k = s
        s = ""
    return k


def xor(a, b):
    ans = ""
    for i in range(len(a)):
```

```
        if a[i] == b[i]:
            ans = ans + "0"
        else:
            ans = ans + "1"
    return ans


initial_perm = [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]


exp_d = [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 ]

per = [ 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 ]


sbox = [[[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] ] ]


final_perm = [ 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 ]

def encrypt(pt, rkb, rk):
    pt = hex2bin(pt)


    pt = permute(pt, initial_perm, 64)
    print("After initial permutation",
bin2hex(pt))


    left = pt[0:32]
    right = pt[32:64]
    for i in range(0, 16):

        right_expanded = permute(right, exp_d, 48)


        xor_x = xor(right_expanded, rkb[i])


        sbox_str = ""
        for j in range(0, 8):
            row = bin2dec(int(xor_x[j * 6] +
xor_x[j * 6 + 5]))
            col = bin2dec(int(xor_x[j * 6 + 1] +
xor_x[j * 6 + 2] + xor_x[j * 6 + 3] + xor_x[j * 6
+ 4]))
            val = sbox[j][row][col]
            sbox_str = sbox_str + dec2bin(val)


        sbox_str = permute(sbox_str, per, 32)


        result = xor(left, sbox_str)
        left = result
```

```
        if(i != 15):
            left, right = right, left
        print("Round", i + 1, ":- ",
bin2hex(left), " ", bin2hex(right), " ", rk[i])


    combine = left + right


    cipher_text = permute(combine, final_perm, 64)
    return cipher_text

print("Data encryption standard (DES):-")

pt = input("Enter the plain text: ")
key = input("Enter the Key: ")


key = hex2bin(key)


keyp = [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 ]

key = permute(key, keyp, 56)

shift_table = [1, 1, 2, 2,
               2, 2, 2, 2,
               1, 2, 2, 2,
               2, 2, 2, 1 ]
```

```
key_comp = [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 ]

left = key[0:28]
right = key[28:56]

rkb = []
rk = []
for i in range(0, 16):

    left = shift_left(left, shift_table[i])
    right = shift_left(right, shift_table[i])


    combine_str = left + right


    round_key = permute(combine_str, key_comp, 48)

    rkb.append(round_key)
    rk.append(bin2hex(round_key))

print("Encryption: ")
cipher_text = bin2hex(encrypt(pt, rkb, rk))
print("Cipher Text : ",cipher_text)

print("Decryption: ")
rkb_rev = rkb[::-1]
rk_rev = rk[::-1]
text = bin2hex(encrypt(cipher_text, rkb_rev,
rk_rev))
```

```
print("Plain Text : ",text)
```

#3DES. WEAK KEYS

2) Try all the weak keys on DES and 3DES, and note the round keys of each Weak key
Ans:In week 4 we have implemented DES
Weak Keys: Weak keys are keys that result in ciphers that are easy to break. If text is encrypted with a weak key, encrypting the resulting cipher again with the same weak key returns the original text.
There are several possibilities like
i) When alternating ones and zeros are given as input.
ii)All zeros appear in the input text
iii)Alternating F , E appear...etc
iv)All zeros
v) All ones..etc;

WEEK-7

#DIFFIE HELMAN :

```
print("Diffie-Hellman Algorithm")
Prime_no = int(input("Enter Prime No.p : "))
g = int(input("Enter Primitive root : "))
PkXa = int(input("Enter Private key of A : "))
PkXb = int(input("Enter Private key of B : "))
ya = g**PkXa % Prime_no
yb = g**PkXb % Prime_no
```

```
ka = yb**PkXa % Prime_no
kb = ya**PkXb % Prime_no
print("Public Key of A: "+str(ya))
print("Public Key of B: "+str(yb))
print("Shared secret key: "+str(ka))
#ELGAMAL CRYPTO
```

Code:

```
import random
from math import pow

a=random.randint(2,10)
def gcd(a,b):
if a<b:
return gcd(b,a)
elif a%b==0:
return b
else:
return gcd(b,a%b)
def gen_key(q):
key= random.randint(pow(10,20),q)
while gcd(q,key)!=1:
key=random.randint(pow(10,20),q)
return key
def power(a,b,c):
x=1
y=a
while b>0:
if b%2==0:
x=(x*y)%c;
y=(y*y)%c
b=int(b/2)
return x%c
def encryption(msg,q,h,g):
ct=[]
k=gen_key(q)
s=power(h,k,q)
```

```
p=power(g,k,q)
for i in range(0,len(msg)):
ct.append(msg[i])
print("g^k used= ",p)
print("g^ak used= ",s)
for i in range(0,len(ct)):
ct[i]=s*ord(ct[i])
return ct,p
def decryption(ct,p,key,q):
pt=[]
h=power(p,key,q)
for i in range(0,len(ct)):
pt.append(chr(int(ct[i]/h)))
return pt
msg= input("Enter message.")

q=random.randint(pow(10,20),pow(10,50))
g=random.randint(2,q)
key=gen_key(q)
h=power(g,key,q)
print("g used=",g)
print("g^a used=",h)
ct,p=encryption(msg,q,h,g)
print("Original Message=",msg)
print("Encrypted Maessage=",ct)
pt=decryption(ct,p,key,q)
d_msg=''.join(pt)
print("Decryted Message=",d_msg)
```

Des using libraries:::

```
pip install pycrypto
pip install base32hex
import base32hex
import hashlib
from Crypto.Cipher import DES
password = "Password"
salt = '\x28\xAB\xBC\xCD\xDE\xEF\x00\x33'
key = password + salt
m = hashlib.md5(key)
key = m.digest()
(dk, iv) =(key[:8], key[8:])
crypter = DES.new(dk, DES.MODE_CBC, iv)

plain_text= "I see you"

print("The plain text is : ",plain_text)
plain_text += '\x00' * (8 - len(plain_text) % 8)
ciphertext = crypter.encrypt(plain_text)
encode_string= base32hex.b32encode(ciphertext)
print("The encoded string is : “,encode_string)

import base32hex
import hashlib
from Crypto.Cipher import DES
password = "Password"
salt = '\x28\xAB\xBC\xCD\xDE\xEF\x00\x33'
key = password + salt
m = hashlib.md5(key)
key = m.digest()
(dk, iv) =(key[:8], key[8:])
crypter = DES.new(dk, DES.MODE_CBC, iv)

encrypted_string='UH562EGM8RCHHTOUC5CTRS59OG======
'

print("The ecrypted string is :
",encrypted_string)
```

```
encrypted_string=base32hex.b32decode(encrypted_str
ing)
decrypted_string =
crypter.decrypt(encrypted_string)
print("The decrypted string is :
",decrypted_string)
```