S.THARUN-19MID0031

1) ADVANCED ENCRYPTION STANDARD

USER DEFINED FUNCITONS

In [1]:

```
def str_to_4x4(text):
    text_lst = []
    for i in text:
        text_lst.append(hex(ord(i))[2:])
    return [[text_lst[j] for j in range(i,16,4)] for i in range(4)]

def return_word(matrix,n):
    return [i[n-1] for i in matrix]

def g(word,n):
    word1 = word.copy()
    temp = word1.pop(0)
    word1.append(temp)
    word1 = check_2dig(word1)
    word1 = [s_box.loc[i[0]][i[1]] for i in word1]
    gword = xor(word1,[rc[n],'00','00','00'])
    return gword

def xor(word1,word2):
    result = []
    for i,j in zip(word1,word2):
        result.append(hex(int('0x'+i,16)^int('0x'+j,16))[2:])
    return result

def check_2dig(word):
    for i in range(len(word)):
        if len(word[i])!=2:
            word[i]="0"+word[i]
    return word

def transpose(matrix):
    return [[matrix[j][i] for j in range(4)] for i in range(4)]

def matrix_xor(matrix1, matrix2):
    return [check_2dig([hex(int('0x'+i,16)^int('0x'+j,16))[2:] for i,j in zip(row1,row2)]) for row1,row2 in zip(matrix1,matrix2)]

def sbox_substitute_matrix(matrix):
    return [[s_box.loc[i[0]][i[1]] for i in word] for word in matrix]

def cyl_rotate(matrix):
    for i in range(4):
        matrix[i] = matrix[i][i:]+matrix[i][:i]
    return matrix

def round_to_8bits(bits):
    while(len(bits))>8:
        irr_poly = "100011011"
        bits = bin(int("0b"+bits,2)^int("0b"+irr_poly,2))[2:]
    while(len(bits)<8):
        bits = '0'+bits
    return bits
```

INPUT

In [2]:

```
input_ = "Two One Nine Two"
input_matrix = str_to_4x4(input_)
input_matrix
```

```
Out[2]: [['54', '4f', '4e', '20'],
         ['77', '6e', '69', '54'],
         ['6f', '65', '6e', '77'],
         ['20', '20', '65', '6f']]
```

KEY

In [3]:

```
key = "Thats my Kung Fu"
key_matrix = str_to_4x4(key)
key_matrix
```

```
Out[3]: [['54', '73', '20', '67'],
         ['68', '20', '4b', '20'],
         ['61', '6d', '75', '46'],
         ['74', '79', '6e', '75']]
```

S-BOX

In [4]:

```
s_box = [['63', '7C', '77', '7B', 'F2', '6B', '6F', 'C5', '30', '01', '67', '2B', 'FE', 'D7', 'AB', '76'],
         ['CA', '82', 'C9', '7D', 'FA', '59', '47', 'F0', 'AD', 'D4', 'A2', 'AF', '9C', 'A4', '72', 'C0'],
         ['B7', 'FD', '93', '26', '36', '3F', 'F7', 'CC', '34', 'A5', 'E5', 'F1', '71', 'D8', '31', '15'],
         ['04', 'C7', '23', 'C3', '18', '96', '05', '9A', '07', '12', '80', 'E2', 'EB', '27', 'B2', '75'],
         ['09', '83', '2C', '1A', '1B', '6E', '5A', 'A0', '52', '3B', 'D6', 'B3', '29', 'E3', '2F', '84'],
         ['53', 'D1', '00', 'ED', '20', 'FC', 'B1', '5B', '6A', 'CB', 'BE', '39', '4A', '4C', '58', 'CF'],
         ['D0', 'EF', 'AA', 'FB', '43', '4D', '33', '85', '45', 'F9', '02', '7F', '50', '3C', '9F', 'A8'],
         ['51', 'A3', '40', '8F', '92', '9D', '38', 'F5', 'BC', 'B6', 'DA', '21', '10', 'FF', 'F3', 'D2'],
         ['CD', '0C', '13', 'EC', '5F', '97', '44', '17', 'C4', 'A7', '7E', '3D', '64', '5D', '19', '73'],
         ['60', '81', '4F', 'DC', '22', '2A', '90', '88', '46', 'EE', 'B8', '14', 'DE', '5E', '0B', 'DB'],
         ['E0', '32', '3A', '0A', '49', '06', '24', '5C', 'C2', 'D3', 'AC', '62', '91', '95', 'E4', '79'],
         ['E7', 'C8', '37', '6D', '8D', 'D5', '4E', 'A9', '6C', '56', 'F4', 'EA', '65', '7A', 'AE', '08'],
         ['BA', '78', '25', '2E', '1C', 'A6', 'B4', 'C6', 'E8', 'DD', '74', '1F', '4B', 'BD', '8B', '8A'],
         ['70', '3E', 'B5', '66', '48', '03', 'F6', '0E', '61', '35', '57', 'B9', '86', 'C1', '1D', '9E'],
         ['E1', 'F8', '98', '11', '69', 'D9', '8E', '94', '9B', '1E', '87', 'E9', 'CE', '55', '28', 'DF'],
         ['8C', 'A1', '89', '0D', 'BF', 'E6', '42', '68', '41', '99', '2D', '0F', 'B0', '54', 'BB', '16']]

for i in range(len(s_box)):
    for j in range(16):
        s_box[i][j] = s_box[i][j].lower()

index = ["0","1","2","3","4","5","6","7","8","9","a","b","c","d","e","f"]

from pandas import DataFrame
s_box = DataFrame(s_box, index = index, columns = index)
s_box
```

Out[4]:

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

ROUND CONSTANT

In [5]:

```
rc = ["01","02","04","08","10","20","40","80","1B","36"]
```

KEY GENERATION

In [6]:

```
round_keys = []
w = []
w.append(return_word(key_matrix,1))
w.append(return_word(key_matrix,2))
w.append(return_word(key_matrix,3))
w.append(return_word(key_matrix,4))
round_keys.append(transpose(key_matrix))
for i in range(10):
    round_key = []
    for j in range(4):
        w1 = w[4*i+j]
        w2 = w[4*i+j+3]
        if j==0:
            w2 = g(w2,i)
        wrd = check_2dig(xor(w2,w1))
        w.append(wrd)
        round_key.append(wrd)
    round_keys.append(round_key)
```

In [7]:

```
round_=0
for i in round_keys:
    print("Round {0:^2} : ".format(round_), end="")
    for j in i:
        for k in j:
            print(k,end=" ")
    print()
    round_+=1
```

```
Round 0  : 54 68 61 74 73 20 6d 79 20 4b 75 6e 67 20 46 75
Round 1  : e2 32 fc f1 91 12 91 88 b1 59 e4 e6 d6 79 a2 93
Round 2  : 56 08 20 07 c7 1a b1 8f 76 43 55 69 a0 3a f7 fa
Round 3  : d2 60 0d e7 15 7a bc 68 63 39 e9 01 c3 03 1e fb
Round 4  : a1 12 02 c9 b4 68 be a1 d7 51 57 a0 14 52 49 5b
Round 5  : b1 29 3b 33 05 41 85 92 d2 10 d2 32 c6 42 9b 69
Round 6  : bd 3d c2 87 b8 7c 47 15 6a 6c 95 27 ac 2e 0e 4e
Round 7  : cc 96 ed 16 74 ea aa 03 1e 86 3f 24 b2 a8 31 6a
Round 8  : 8e 51 ef 21 fa bb 45 22 e4 3d 7a 06 56 95 4b 6c
Round 9  : bf e2 bf 90 45 59 fa b2 a1 64 80 b4 f7 f1 cb d8
Round 10 : 28 fd de f8 6d a4 24 4a cc c0 a4 fe 3b 31 6f 26
```

KEY MATRICES

In [8]:

```
key_matrices = []
for i in round_keys:
    key_matrices.append(transpose(i))
Round = 0
for i in key_matrices:
    print("Round", Round)
    for j in i:
        print(j)
    Round = Round+1
    print()
```

```
Round 0
['54', '73', '20', '67']
['68', '20', '4b', '20']
['61', '6d', '75', '46']
['74', '79', '6e', '75']

Round 1
['e2', '91', 'b1', 'd6']
['32', '12', '59', '79']
['fc', '91', 'e4', 'a2']
['f1', '88', 'e6', '93']

Round 2
['56', 'c7', '76', 'a0']
['08', '1a', '43', '3a']
['20', 'b1', '55', 'f7']
['07', '8f', '69', 'fa']

Round 3
['d2', '15', '63', 'c3']
['60', '7a', '39', '03']
['0d', 'bc', 'e9', '1e']
['e7', '68', '01', 'fb']

Round 4
['a1', 'b4', 'd7', '14']
['12', '68', '51', '52']
['02', 'be', '57', '49']
['c9', 'a1', 'a0', '5b']

Round 5
['b1', '05', 'd2', 'c6']
['29', '41', '10', '42']
['3b', '85', 'd2', '9b']
['33', '92', '32', '69']

Round 6
['bd', 'b8', '6a', 'ac']
['3d', '7c', '6c', '2e']
['c2', '47', '95', '0e']
['87', '15', '27', '4e']

Round 7
['cc', '74', '1e', 'b2']
['96', 'ea', '86', 'a8']
['ed', 'aa', '3f', '31']
['16', '03', '24', '6a']

Round 8
['8e', 'fa', 'e4', '56']
['51', 'bb', '3d', '95']
['ef', '45', '7a', '4b']
['21', '22', '06', '6c']

Round 9
['bf', '45', 'a1', 'f7']
['e2', '59', '64', 'f1']
['bf', 'fa', '80', 'cb']
['90', 'b2', 'b4', 'd8']

Round 10
['28', '6d', 'cc', '3b']
['fd', 'a4', 'c0', '31']
['de', '24', 'a4', '6f']
['f8', '4a', 'fe', '26']
```

ADD ROUND KEYS

In [9]:

```
state_matrices = []
state_matrices.append(input_matrix)
```

In [10]:

```
mix_col=[['02','03','01','01'],
         ['01','02','03','01'],
         ['01','01','02','03'],
         ['03','01','01','02']]
```

In [11]:

```
for k in range(1,10):
    f = transpose(cyl_rotate(sbox_substitute_matrix(matrix_xor(state_matrices[k-1],key_matrices[k-1]))))
    round_key_mat = []
    for row1 in mix_col:
        temp_row = []
        for row2 in f:
            temp=0
            for i,j in zip(row1,row2):
                m = int('0x'+i,16)
                n = int('0x'+j,16)
                if m==3:
                    temp = temp^int("0b"+round_to_8bits(bin((n<<1)^n)[2:]),2)
                else:
                    temp = temp^int("0b"+round_to_8bits(bin(n*m)[2:]),2)
            temp_row.append(hex(int("0b"+round_to_8bits(bin(temp)[2:]),2))[2:])
        round_key_mat.append(temp_row)
    print("Round",k)
    state_matrices.append(round_key_mat)
    for a in matrix_xor(round_key_mat,key_matrices[k]):
        print(a)
    print()
f = cyl_rotate(sbox_substitute_matrix(matrix_xor(state_matrices[k],key_matrices[k])))
print("Round 10")
cipher_mat = matrix_xor(f,key_matrices[10])
for i in cipher_mat:
    print(i)

print()
print("Cipher Text ==>"," ".join([" ".join(i) for i in transpose(cipher_mat)]))
```

```
Round 1
['58', '15', '59', 'cd']
['47', 'b6', 'd4', '39']
['08', '1c', 'e2', 'df']
['8b', 'ba', 'e8', 'ce']

Round 2
['43', '0e', '09', '3d']
['c6', '57', '08', 'f8']
['a9', 'c0', 'eb', '7f']
['62', 'c8', 'fe', '37']

Round 3
['78', '70', '99', '4b']
['76', '76', '3c', '39']
['30', '7d', '37', '34']
['54', '23', '5b', 'f1']

Round 4
['b1', '08', '04', 'e7']
['ca', 'fc', 'b1', 'b2']
['51', '54', 'c9', '6c']
['ed', 'e1', 'd3', '20']

Round 5
['9b', '23', '5d', '2f']
['51', '5f', '1c', '38']
['20', '22', 'bd', '91']
['68', 'f0', '32', '56']

Round 6
['14', '8f', 'c0', '5e']
['93', 'a4', '60', '0f']
['25', '2b', '24', '92']
['77', 'e8', '40', '75']

Round 7
['53', '43', '4f', '85']
['39', '06', '0a', '52']
['8e', '93', '3b', '57']
['5d', 'f8', '95', 'bd']

Round 8
['66', '70', 'af', 'a3']
['25', 'ce', 'd3', '73']
['3c', '5a', '0f', '13']
['74', 'a8', '0a', '54']

Round 9
['09', 'a2', 'f0', '7b']
['66', 'd1', 'fc', '3b']
['8b', '9a', 'e6', '30']
['78', '65', 'c4', '89']

Round 10
['29', '57', '40', '1a']
['c3', '14', '22', '02']
['50', '20', '99', 'd7']
['5f', 'f6', 'b3', '3a']

Cipher Text ==> 29 c3 50 5f 57 14 20 f6 40 22 99 b3 1a 02 d7 3a
```

2) DIFFIE HELLMAN KEY EXCHANGE

In [10]:

```
from math import sqrt
from numpy import unique

def isPrime(num):
    for i in range(2,int(sqrt(num))):
        if num%i == 0:
            return False
    return True

def isPrimitiveRoot(p,g):
    row = []
    for i in range(1,p):
        row.append((g**i)%p)
    if len(unique(row)) == len(row):
        return True
    return False
```

GLOBAL VARIABLES

In [14]:

```
p = int(input("Enter p : "))
g = int(input("Enter p : "))
while not isPrimitiveRoot(p,g) and g<p:
    print("Oops...g must me primitive root of P. Try again...")
    p = int(input("Enter p : "))
    g = int(input("Enter p : "))
```

```
Enter p : 61
Enter p : 6
```

Choose private Key

In [15]:

```
Xa = int(input("Alice private Key : "))
Xb = int(input("Bob private Key   : "))
```

```
Alice private Key : 50
Bob private Key   : 39
```

Calculate public key

In [16]:

```
Ya = (g**Xa)%p
Yb = (g**Xb)%p
```

Calculate shared secret

In [19]:

```
K1 = (Yb**Xa)%p
K2 = (Ya**Xb)%p
if K1 == K2:
    print("Secret key of Alice ==>",K1)
    print("Secret key of Bob   ==>",K2)
    print("Secret key of Alice and Bob are same")
else:
    print("Key sharing Unsuccessful")
```

```
Secret key of Alice ==> 60
Secret key of Bob   ==> 60
Secret key of Alice and Bob are same
```

3) RSA ALGORITHM

User Defined Functions

In [2]:

```
from numpy import gcd
from math import sqrt

def isPrime(n):
    for i in range(2,int(sqrt(n))):
        if n%i == 0:
            return False
    return True
```

Selecting two large prime numbers

and Apply Euler totient function

In [4]:

```
p=int(input("Enter p : "))
while not isPrime(p):
    p=int(input("P should be prime number..Try again..Enter p : "))
q=int(input("Enter q : "))
while not isPrime(q):
    q=int(input("Q should be prime number..Try again..Enter q : "))
n = p*q
phi_n = (p-1)*(q-1)
```

```
Enter p : 3
Enter q : 5
```

Calculate Public Key

In [5]:

```
for e in range(2,phi_n+1):
    if gcd(e,phi_n) == 1:
        break
```

Calculate Private Key

In [6]:

```
d=1
while True:
    if (d*e)%phi_n == 1:
        break
    d=d+1
```

Plain Text

In [7]:

```
M = int(input("Enter plain_text : "))
while(M>n):
    M = int(input("Enter plain_text less than "+str(n)+" : "))
```

```
Enter plain_text : 4
```

Encryption and Decryption

In [8]:

```
# Encryption
C = (M**e)%n

# Decryption
M = (C**d)%n

print("Encrypted Cipher Text :",C)
print("Decrypted Plain Text  :",M)
```

```
Encrypted Cipher Text : 4
Decrypted Plain Text  : 4
```

4) ELGAMAL ALGORITHM

GLOBAL VARIABLES

In [11]:

```
p = int(input("Enter p : "))
g = int(input("Enter p : "))
while not isPrimitiveRoot(p,g) and g<p:
    print("Oops...g must me primitive root of P. Try again...")
    p = int(input("Enter p : "))
    g = int(input("Enter p : "))
```

```
Enter p : 61
Enter p : 6
```

Sender side keys

In [12]:

```
# ALICE
print("Hey Alice...")
Xa = int(input("Enter Private Key : ")) # 50
Ya = int(input("Enter Public Key  : ")) # 14
```

```
Hey Alice...
Enter Private Key : 50
Enter Public Key  : 14
```

Receiver side keys

In [13]:

```
# BOB
print("Hey Bob...")
Xb = int(input("Enter Private Key : ")) # 39
Yb = int(input("Enter Public Key  : ")) # 53
```

```
Hey Bob...
Enter Private Key : 39
Enter Public Key  : 53
```

Plain Text

In [14]:

```
M=int(input("Enter plain_text : ")) # 4
```

```
Enter plain_text : 4
```

Encryption and Decryption

In [15]:

```
# Encryption
Yenc = (M*(Ya**Xb))%p

# Decryption
Pln_txt = ((Yenc%p) * (Yb**(p-1-Xa))%p)%p

print("Encrypted Cipher Text :",Yenc)
print("Decrypted Plain Text  :",Pln_txt)
```

```
Encrypted Cipher Text : 57
Decrypted Plain Text  : 4
```