DES, CRT and EEA
Cryptography – Lab 2

Name: Ojas Patil
Reg No: 21BAI1106

Theorems Covered

1. Data Encryption Standard
2. Chinese Remainder Theorem
3. Extended Euclidean Algorithm

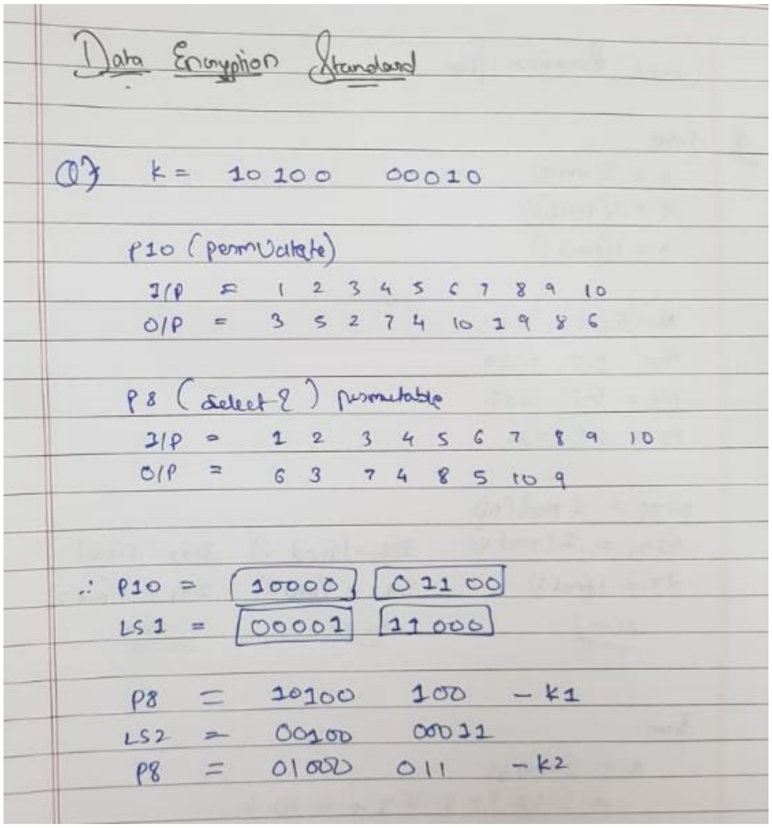
DES - Data Encryption Standards

Output Snapshot

Command Prompt

```
C:\VIT\sem 7\crypto lab\lab2>python des.py
Enter the key bits (56 bits) seperated by space 1 0 0 0 1 0 1 1 1 0 0 1 0 1 0
 0 0 0 1 1 0 0 0 1 1 1 0 1 0 1 1 0 1 0 1 0 0 1 0 1 0 0 0 1 1 0 1 0 0 1 0 0 1
0 0 0
Enter the data bits (64) to encrypt or decrypt seperated by space 1 0 0 0 1 0
 1 1 1 0 0 1 0 1 0 0 0 0 1 1 0 0 0 1 1 1 0 1 0 1 1 0 1 0 1 0 0 1 0 1 0 0 0 1
1 0 1 0 0 1 0 0 1 0 0 0 1 1 0 1 0 0 1 0
data entry accepted, data loaded succesfully
key entry accepted, key loaded succesfully
Choose 0 for encryption or Choose 1 for decryption 0
Time taken to encrypt the data with DES is 0.015886306762695312
Encrypted data is [1 0 0 0 0 0 0 1 1 1 1 0 0 0 1 0 1 0 1 0 0 0 0 0 0 0 0 0 0
1 1 0 1 1 1 1 1
 1 0 0 1 0 0 1 1 1 0 1 0 1 1 1 1 0 1 0 1 1 0 0 1 1 1 1]

C:\VIT\sem 7\crypto lab\lab2>
```

Handwritten sum



Data Encryption Standard

Q) k = 10100 00010

P10 (permutate)

I/P = 1 2 3 4 5 6 7 8 9 10

O/P = 3 5 2 7 4 10 1 9 8 6

P8 (select 8) permutable

I/P = 1 2 3 4 5 6 7 8 9 10

O/P = 6 3 7 4 8 5 10 9

∴ P10 = $\boxed{10000}$ $\boxed{01100}$

LS1 = $\boxed{00001}$ $\boxed{11000}$

P8 = 10100 100 - k1

LS2 = 00100 00011

P8 = 01000 011 - k2

Source Code

```
# encryption algorithm
import numpy as np
import time

IP = [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]

FP = [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]

EBox = [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]

SBox =[
        # S1
        [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],

        # S2
        [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],

        # S3
        [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],

        # S4
        [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],

        # S5
        [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],
```

```python
        # S6
        [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],

        # S7
        [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],

        # S8
        [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],
    ]

F_PBox = [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 ]

key_PBox = [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]

def xor(left,xorstream):
    xorresult = np.logical_xor(left,xorstream)

    xorresult  = xorresult.astype(int)

    return xorresult

def E_box(right):
    expanded = np.empty(48)
    j = 0
    for i in EBox:
        expanded[j] = right[i - 1]
```

```
        j += 1
    expanded = list(map(int,expanded))
    expanded = np.array(expanded)
    return expanded

#clean this code please (sboxlookup)
def sboxloopup(sinput,x):
    tableno = x - 1
    row = int((np.array2string(sinput[0]) + np.array2string(sinput[5])),2)

    # make this part of the code better
    column = sinput[1:5]
    column = np.array2string(column)
    column = column[1:8].replace(" ", "")
    column = int(column,2)

    elementno = (16 * row) + column
    soutput = SBox[tableno][elementno]
    soutput = list(np.binary_repr(soutput, width=4))
    soutput= np.array(list(map(int, soutput)))
    return soutput

def sbox(sboxin):
#takes 48 bit input and return 32 bit
    sboxin1 = sboxin[0:6]
    sboxout1 = sboxloopup(sboxin1,1)
    sboxin2 = sboxin[6:12]
    sboxout2 = sboxloopup(sboxin2,2)
    sboxin3 = sboxin[12:18]
    sboxout3 = sboxloopup(sboxin3, 3)
    sboxin4 = sboxin[18:24]
    sboxout4 = sboxloopup(sboxin4, 4)
    sboxin5 = sboxin[24:30]
    sboxout5 = sboxloopup(sboxin5, 5)
    sboxin6 = sboxin[30:36]
    sboxout6 = sboxloopup(sboxin6, 6)
    sboxin7 = sboxin[36:42]
    sboxout7 = sboxloopup(sboxin7, 7)
    sboxin8 = sboxin[42:48]
    sboxout8 = sboxloopup(sboxin8, 8)
    sboxout =
np.concatenate([sboxout1,sboxout2,sboxout3,sboxout4,sboxout5,sboxout6,sboxout7,sb
oxout8])
    return sboxout
```

```
def f_permute(topermute):
    permuted= np.empty(32)
    j = 0
    for i in F_PBox:
        permuted[j] = topermute[i - 1]
        j += 1
    return permuted

def f_function(right,rkey):
    expanded = E_box(right)
    xored = xor(expanded,rkey)
    sboxed = sbox(xored)
    xorstream = f_permute(sboxed)
    return xorstream

def round(data,rkey):
    l0 = data[0:32]
    r0 = data[32:64]
    xorstream = f_function(r0,rkey)
    r1 = xor(l0,xorstream)
    l1 = r0
    returndata = np.empty_like(data)
    returndata[0:32] = l1
    returndata[32:64] = r1
    return(returndata)

def permutation(data,x):
    #intial and final permutation conditional based on other passed value
    permute1 = np.empty_like(IP)
    if x == 0:
        j = 0
        for i in IP:
            permute1[j] = data[i-1]
            j += 1
        return(permute1)
    else:
        permute2 = np.empty_like(FP)
        k = 0
        for l in FP:
            permute2[k] = data[l-1]
            k += 1
        return(permute2)

def userinput():
```

```
    keyinp = input("Enter the key bits (56 bits) seperated by space "
"").strip().split()
    datainp = input("Enter the data bits (64) to encrypt or decrypt seperated by
space " "").strip().split()
    #change to 56 later
    lenofkey = 56
    #change to 64 later
    lenofdata = 64
    if len(datainp) == lenofdata and len(keyinp) == lenofkey:
        print("data entry accepted, data loaded succesfully")
        print("key entry accepted, key loaded succesfully")
    else:
        while len(datainp) != lenofdata:
            print("length of data entered ",len(datainp))
            datainp = input("Error in entered data. Enter the data (64 bits) to
encrypt or decrypt seperated by space " "").strip().split()

        print("data entry accepted, data loaded succesfully")
        while len(keyinp) != lenofkey:
            print("length of key entered ", len(keyinp))
            keyinp = input("Error in entered key. Enter the key (56 bits) to
encrypt or decrypt seperated by space " "").strip().split()
        print("key entry accepted, key loaded succesfully")
#also add functionality to accept 64 bit keys instead of 54
    return keyinp,datainp

def keyshift(toshift,n):
    if (n == 1) or (n == 2) or (n == 9) or (n == 16):
        toshift= np.roll(toshift,-1)
        return toshift
    else:
        toshift = np.roll(toshift, -2)
        return toshift

def keypermute(key16):
    keypermuted = np.empty([16,48])
    l = 0
    for k in key16:
        j = 0
        for i in key_PBox:
            keypermuted[l][j] = k[i - 1]
            j += 1
        l += 1
    return keypermuted
```

```python
#
def keyschedule(key):
    left = key[0:28]
    right = key[28:56]
    shifted = np.zeros(56)
    key16 = np.zeros([16,56])
    for i in range(1,17):
        shifted[0:28] = keyshift(left,i)
        shifted[28:56] = keyshift(right,i)
        left = shifted[0:28]
        right = shifted[28:56]
#add shifted to key16 and return key16
        key16[i - 1] = shifted
#key16 is the final shifted 16 key pair now to permute
    key16 = keypermute(key16)
    key16 = [list(map(int, x)) for x in key16]
    key16 = np.array(key16)
    return key16

def main():
    key, data = userinput()
    # key = ['1', '1', '1', '1', '1', '1', '1', '0', '0', '1', '1', '1', '0',
'0', '0', '1', '1', '1', '0', '0', '1', '1', '1', '0', '0', '0', '1', '1', '1',
'0', '0', '0', '0', '1', '1', '1', '0', '0', '0', '0', '1', '1', '1', '0', '0',
'0', '1', '1', '1', '0', '0', '0', '0', '0', '0', '0']
    # data = ['1', '1', '1', '1', '1', '1', '1', '0', '0', '1', '1', '1', '0',
'0', '0', '1', '1', '1', '0', '0', '1', '1', '1', '0', '0', '0', '1', '1', '1',
'0', '0', '0', '0', '1', '1', '1', '0', '0', '0', '0', '1', '1', '1', '0', '0',
'0', '1', '1', '1', '0', '0', '0', '0', '0', '0', '0', '0', '1', '0', '1', '1',
'0', '1', '0']
    # # print(data,key)
#taking input for decryption and encryption
    operate = int(input("Choose 0 for encryption or Choose 1 for decryption "))
    starttime = time.time()
    key16 = keyschedule(key)

    if operate == 0:
        data = permutation(data,0)
# testing round function now
        for i in range(16):
            data = round(data,key16[i])

#making left side right and right side left
        data = np.roll(data,32)
        data = (permutation(data, 1))
```

```
        print("Time taken to encrypt the data with DES is", time.time() - 
starttime)
        print("Encrypted data is", data)

    if operate == 1:
        data = permutation(data, 0)
        # testing round function now
        for i in range(16):
            data = round(data, key16[16 - (i + 1)])

        data = np.roll(data, 32)
        data = (permutation(data, 1))
        print("Time taken to decrypt the data with DES is", time.time() - 
starttime)
        print("Decrypted data is", data)

main()

#data to test
#key = 1 0 0 0 1 0 1 1 1 0 0 1 0 1 0 0 0 0 1 1 0 0 0 1 1 1 0 1 0 1 1 0 1 0 1 0 0 
1 0 1 0 0 0 1 1 0 1 0 0 1 0 0 1 0 0 0
# #data = 1 0 0 0 1 0 1 1 1 0 0 1 0 1 0 0 0 0 1 1 0 0 0 1 1 1 0 1 0 1 1 0 1 0 1 0 
0 1 0 1 0 0 0 1 1 0 1 0 0 1 0 0 1 0 0 0 1 1 0 1 0 0 1 0
```

CRT - Chinese Remainder Theorem

Output Snapshot

```
Select Command Prompt

C:\VIT\sem 7\crypto lab\lab2>python crt.py
[*] p = 691
[*] q = 701
[*] n = p * q = 484391
[*] phi = 483000
[*] e = 17
[*] d = 369353
[*] m = 72345
[*] c = 246536
[*] test: c = 72345**e % n = 246536**d % n = m_d = 72345
[*] enc expr:
 python -m timeit -s 'm = 72345; e = 17; n = 484391'  'c = (m**e) % n'
-n 100
[*] dec expr:
```

```
def mulinv(b, n):
    g, x, _ = egcd(b, n)
    if g == 1:
        return x % n

qInv = mulinv(701, 691)
m_g = ( ( ( (481-142)*qInv ) % 691) * 701 ) + 142'

C:\VIT\sem 7\crypto lab\lab2>
```

Handwritten Sum

Chinese Remainder Theorem

Q. Solve

$x = 3 \pmod 5$

$x = 2 \pmod 6$

$x = 1 \pmod 7$

$N = 5 \cdot 6 \cdot 7 = 210$

$N_1 = 6 \cdot 7 = 42$

$N_2 = 5 \cdot 7 = 35$

$N_3 = 6 \cdot 5 = 30$

$N_i x_i = 1 \bmod (n_i)$

$42 x_1 = 1 \pmod 5$ $35 x_2 = 1 \pmod 6$ $30 x_3 = 1 \pmod 7$

$2 x_1 = 1 \pmod 5$ $5 x_2 = 1 \pmod 6$ $2 x_3 = 1 \pmod 7$

$x_1 = 3$ $x_2 = 5$ $x_3 = 4$

Form:

$a = \sum x_i N_i b_i$

$= 3 \cdot 42 \cdot 3 + 5 \cdot 35 \cdot 2 + 4 \cdot 30 \cdot 1$

$= 848$

$\approx 8 \pmod{210}$

Source Code

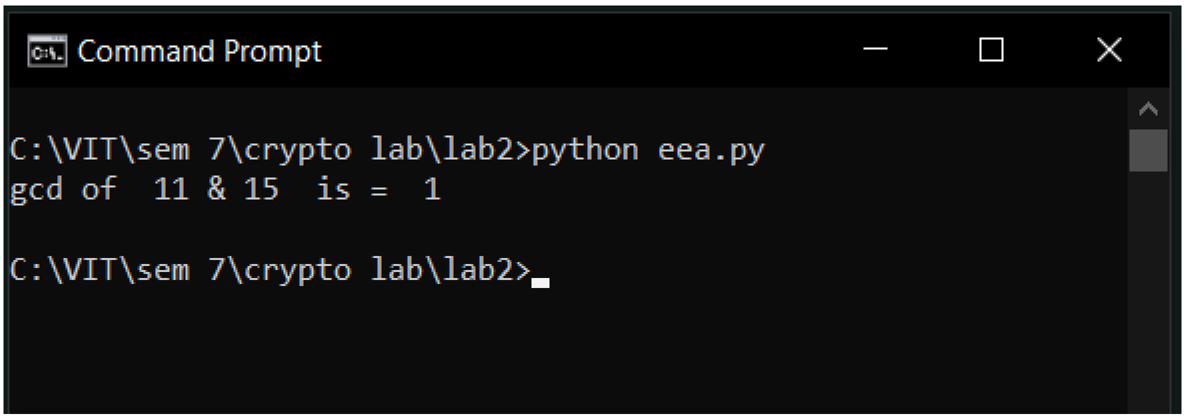
```
import sys
import os
import time
import timeit
import sys

sys.setrecursionlimit(1000000)

p = 691
q = 701
print("[*] p = %d" % (p))
print("[*] q = %d" % (q))

n = p*q
print("[*] n = p * q = %d" % (n))

phi = (p-1)*(q-1)
print("[*] phi = %d" % (phi))

e = 17
print("[*] e = %d" % (e))

def egcd(a, b):
    if a == 0:
        return (b, 0, 1)
    else:
        g, x, y = egcd(b % a, a)
        return (g, y - (b // a) * x, x)

def mulinv(b, n):
    g, x, _ = egcd(b, n)
    if g == 1:
        return x % n

d = mulinv(e, phi)
print("[*] d = %d" % (d))

m = 72345
print("[*] m = %d" % (m))

pub_k = (e, n)
priv_k = (d, n)
```

```
# Let's perform encryption of a message m using the public key (e,n)
c = (m**e) % n
print("[*] c = %d" % (c))
m_d = (c**d) % n
print("[*] test: c = %d**e %% n = %d**d %% n = m_d = %d" % (m, c, m_d))
print("[*] enc expr:\n python -m timeit -s 'm = %d; e = %d; n = %d'  'c = (m**e)
%% n' -n 100" % (m, e, n))
print("[*] dec expr:\n python -m timeit -s 'c = %d; d = %d; n = %d'  'm = (c**d)
%% n' -n 100" % (c, d, n))

a1 = c**d % p
a2 = c**d % q
print("[*] a1= %d, a2 = %d" % (a1, a2))
print("[*] expr a1:\n python -m timeit -s 'c = %d; d = %d; p = %d'  '(c**d) %%
p'" % (c, d, p))
print("[*] expr a2:\n python -m timeit -s 'c = %d; d = %d; q = %d'  '(q**d) %%
q'" % (c, d, q))

# Next, let's reduce both bases c by their respective moduli:
a1 = ((c%p)**d) % p
a2 = ((c%q)**d) % q
print("[*] a1= %d, a2 = %d" % (a1, a2))
print("[*] expr a1 w/ reduction of c:\n python -m timeit -s 'c = %d; d = %d; p =
%d'  '((c%%p)**d) %% p'" % (c, d, p))
print("[*] expr a2 w/ reduction of c:\n python -m timeit -s 'c = %d; d = %d; q =
%d'  '((c%%q)**d) %% q'" % (c, d, q))

dP = d % (p-1)
dQ = d % (q-1)
a1 = ((c%p)**dP) % p
a2 = ((c%q)**dQ) % q
print("[*] a1= %d, a2 = %d" % (a1, a2))
print("[*] expr a1 w/ reduction of c:\n python -m timeit -s 'c = %d; dP = %d; p =
%d'  '((c%%p)**dP) %% p'" % (c, dP, p))
print("[*] expr a2 w/ reduction of c:\n python -m timeit -s 'c = %d; dQ = %d; q =
%d'  '((c%%q)**dQ) %% q'" % (c, dQ, q))
qInv = mulinv(q, p)
print("[*] qInv = %d" % (qInv))

m_g = ( ( ( (a1-a2)*qInv ) % p) * q ) + a2
print("[*] m = %d, m_d = %d, m_g = %d" % (m, m_d, m_g))

# Print the code for timing the execution of the Garner's formula to reconstruct
the actual message m from a1 and a2.
```

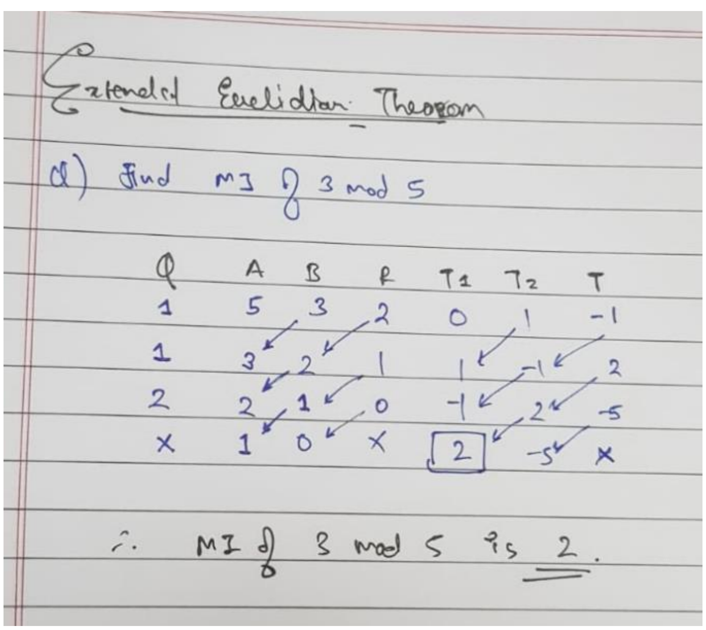
```
print("[*] expr a1 w/ reduction of c:\n python -m timeit -s '\ndef egcd(a, b):\n    if a == 0:\n        return (b, 0, 1)\n    else:\n        g, x, y = egcd(b %% a, a)\n        return (g, y - (b // a) * x, x)\n\n\ndef mulinv(b, n):\n    g, x, _ = egcd(b, n)\n    if g == 1:\n        return x %% n\n\nqInv = mulinv(%d, %d)\nm_g = ( ( ( (%d-%d)*qInv ) %% %d) * %d ) + %d'" % (q, p, a1, a2, p, q, a2))
```

EEA - Extended Euclidean Algorithm

Output Snapshot



Handwritten Sum



Source Code

```
# extended Euclidean Algorithm
def gcdExtended(a, b, x, y):
   # Base Case
   if a == 0 :
      x = 0
      y = 1
      return b
   x1 = 1
   y1 = 1 # storing the result
   gcd = gcdExtended(b%a, a, x1, y1)
   # Update x and y with previous calculated values
   x = y1 - (b/a) * x1
   y = x1
   return gcd

x = 1
y = 1
a = 11
b = 15
g = gcdExtended(a, b, x, y)
print("gcd of ", a , "&" , b, " is = ", g)
```

Conclusion

We explored the Data Encryption Standard (DES), the Chinese Remainder Theorem (CRT), and the Extended Euclidean Algorithm, understanding their roles in encryption, decryption, and mathematical operations. Through Python examples, we demonstrated how these concepts secure information and perform essential calculations in cryptography.