HW4_EE_4940_code

October 26, 2024

[1]:
```
#Import libraries
import sympy as sp
import numpy as np
from decimal import *
np.set_printoptions(legacy='1.25')
```

0.1 Problem 4

[2]:
```
def RSA_encrypt(M,e,n) : #Encypts a message using RSA.
    """ Inputs:
            M: 8 digit number representation of a 4 letter word.
            e: RSA public key list, e.g. 17 = [1,0,0,0,1,1,0,0,0,0]
            n: n in RSA, public information.
        Returns: M^e (mod n)
    """
    c = 0
    f = 1
    for i in e :
        c = c*2
        f = f*f % n
        if i==1 :
            c = c+1
            f = np.multiply(f,M) % n

    return f
```

[3]:
```
# A quick example from lecture to make sure my algorithm works.
n = 561
e = [1,0,0,0,1,1,0,0,0,0]
M = 7
lecture_example = RSA_encrypt(7,e,n)
print(f'Expected Value = 1; Actual Value = {lecture_example}')
```

```
Expected Value = 1; Actual Value = 1
```

[4]:
```
#Problem 4 work.
n=121643759
e = [0,0,0,1,0,0,0,1] #e = 17
```

```
M1 = 3414142 #'Coop'
M2 = 31440006 #'er F'
M3 = 44413139 #'roem'
M4 = 39354033 #'ming'

C1 = RSA_encrypt(M1,e,n)
C2 = RSA_encrypt(M2,e,n)
C3 = RSA_encrypt(M3,e,n)
C4 = RSA_encrypt(M4,e,n)
print(f'C1 = 00{C1}, C2 = {C2}, C3 = {C3}, C4 = {C4}')
```

```
C1 = 00201088, C2 = 85403970, C3 = 19023410, C4 = 22989214
```

0.2 Problem 5

[5]:
```
#Must have R = [], and Q = [] set outside of function
def euclidean_algorithm(a,b,R,Q) :
    """ Recursive function to
        find remainders in euclidean algorithm.

        Inputs:
        a,b: integers in euclidean algorithm.
        R is empty list for remainders to be appended
        Q is empty list for quotients to be appended

        Returns: current remainder.
    """
    q = int(a/b)
    r = a-b*q
    R.append(r)
    Q.append(q)
    if r == 1 : # Base Case
        return
    euclidean_algorithm(b,r,R,Q)


def euclidean_mult_inverses(a,b) :
    """ Uses euclidean algorithm to determine
        x,y that satisfies x*a + y*b = 1

        Returns: [x,y]
    """
    R = []
    Q = []
    euclidean_algorithm(a,b,R,Q)
    A = [[0,len(Q)-x-1] for x in range(0,len(Q))]
    R.reverse()
    Q.reverse()
```

```
    A[0][0] = -Q[0]
    A[1][0] = 1
    m1 = 0
    m2 = 0
    i=1
    while i<len(R) :
        #print(f'i={i}')
        A_temp = A[i-1][0]
        #print(f'A_temp = {A_temp}')
        #print(f'Q[{i}] = {Q[i]}')
        A[i-1][0] = 0
        A[i][0] = A[i][0] - A_temp*Q[i]
        #print(A)
        if (i+1)<len(R) : A[i+1][0] = A[i+1][0] + A_temp
        else : x = A_temp
        i+=1
    y = A[len(A)-1][0]
    #print(f'm1={x}, m2={y}')
    return [x,y]

def find_CRT_remainder(list) :
    """ Inputs:
                list: [[n1,a1],[n2,a2],...[nk,ak]]
        Returns a "total" that is satisfies
                        total (mod n1) = a1,
                        total (mod n2) = a2,
                        ... ,
                        total (mod nk) = ak.
    """
    #Check if all numbers are coprime
    for set1 in list :
        for set2 in list :
            if set1 != set2 :
                gcd = np.gcd(set1[0],set2[0])
                if (gcd != 1) :
                    print(f'ERROR: {set1[0]} and {set2[0]} not GCD')
                    return -1
    sum = [0 for x in range(0,len(list))]
    #Find Capital M
    i = 0
    for n_a_pair in list :
        M = 1
        for m_pair in list :
            if (m_pair != n_a_pair) : M = M*m_pair[0] #M1 = n2*n3*n4*...*nk, M2␣
↪= n1*n3*n4*...*nk, etc.
        A = euclidean_mult_inverses(M,n_a_pair[0])[0] # Ai, solution for␣
↪AiMi+Bi*ni = 1
```

```
        sum[i] = A*n_a_pair[1]*M #Ai*Mi*ai
        i+=1
    total = 0
    print(sum)
    for num in sum :
        total = total + num
    return total

def decimal_cuberoot(input) :
    """ input: integer value

        Returns: cube root of input.
    """
    x = str(input)
    minprec = 40
    if len(x) > minprec: getcontext().prec = len(x)
    else:                getcontext().prec = minprec

    x = Decimal(x)
    power = Decimal(1)/Decimal(3)

    answer = x**power
    ranswer = answer.quantize(Decimal('1.'), rounding=ROUND_UP)

    #diff = x - ranswer**Decimal(3)
    # if diff == Decimal(0):
    #     print("x is the cubic number of", ranswer)
    # else:
    #     print("x has a cubic root of ", answer)

    return answer
```

[6]:
```
print('Checking if euclidean_mult_inverses works...')
print(f'Expected = [102,-209], Result = {euclidean_mult_inverses(543,265)}')␣
↪#from P5a
print(f'Expected = [-32,17739], Result = {euclidean_mult_inverses(54880,99)}')␣
↪#from P3
```

```
Checking if euclidean_mult_inverses works…
Expected = [102,-209], Result = [102, -209]
Expected = [-32,17739], Result = [-32, 17739]
```

[7]:
```
#Problem 5 d)
nA = 64652191
nB = 53275609
```

```
nC = 67903951
CA = 2461786
CB = 32328918
CC = 40602713
input = [[nA,CA],[nB,CB],[nC,CC]]

output = find_CRT_remainder(input)
M = decimal_cuberoot(output)

print(f'output = {output}')
print(f'output % nA = {output % nA}')
print(f'output % nB = {output % nB}')
print(f'output % nC = {output % nC}')
print(f'M = {M}')
```

```
[-63260066211410278583287016O2, -240626162073953524117964624O586,
31264214177657889879000341558O5]
output = 713833790405112718862059213617
output % nA = 2461786
output % nB = 32328918
output % nC = 40602713
M = 8937149730.248965038069054102853901352268
```

[8]:
```
#Check to see if root3 is actually the cubed root.
print(M ** 3) #It is!
```

```
713833790405112718862059213616.9999999950
```