Menu

August 26, 2024

[2]:
```python
def Convert_Binary_String(_int):
    return bin(_int)[2:]

def FME(b, n, m):
    x = 1 #stores final result
    power = b % m
    binary = Convert_Binary_String(n) #get power in binary,
    binary = binary[::-1] # reverse it to iterate right to left
    for i in range(len(binary)): #for loop to run through each bit in binary␣
↪string
        if binary[i] == '1': #if bit is 1
            x = (x*power) % m   #update result with current power
        power = (power*power) % m #update power for next interation
    return x #final result

def Euclidean_Alg(a, b):
    while b != 0:
        remainder = a % b #calculate remainder of a/b
        a = b
        b = remainder
    return a #when b is 0, a is GCD and returns

def EEA(a, b):
    #check for pos. int.

    if a <= 0 or b <=0:
        print("Error, inputs must be positive integers")
        return

    #initalize variables
    inital_a = a
    inital_a = b
    (s1, t1) = (1, 0) #coeff. for a
    (s2, t2) = (0, 1) #coeff. for b

    while b > 0: #loop until remainder is 0
        k = a % b #remainder
```

```python
        q = a // b #integer division to get quotient

        #update m and n
        a = b
        b = k

        (s1H, t1H) = (s2,t2) #"bookkeep to hold current s2,t2"
        (s2H, t2H) = (s1 - q*s2, t1 - q*t2) #update using bezout coeff. equation

        #move current coeffs. to next interation
        (s1,t1) = s1H,t1H
        (s2,t2) = s2H,t2H

    return a,(s1,t1)

def Find_Public_Key_e(p, q):
    n = p*q
    rp = (p-1)*(q-1) #rp is what e should be relatively prime to
    e = 3 #start with small prime number
    while Euclidean_Alg(e, rp)!=1 & e!=p & e!=q: #loop incrementing by 2 for odd numbers, to look for relatively prime e
        e += 2
    return (n,e)

def Find_Private_Key_d(e, p, q):
    rp = (p-1)*(q-1)
    gcd, (s1,t1) = EEA(e, rp) #EEA to find gcd, and coeffs.

    if gcd!=1: #if gcd is not 1, e and rp aren't co prime
        print("No modular inverse exits")
        return

    d = s1 % rp #x is modular inverse of e

    if d<0: #if d isn't positive, add rp
        d+= rp
    return d

def Convert_Text(_string):
    integer_list = []
    for char in _string:
        integer_list.append(ord(char)) #run through string as a list and convert each char to ascii then add to integer list
    return integer_list

def Convert_Num(_list):
    _string = ''
```

```python
    for i in _list:
        _string += chr(i)
    return _string

def Encode(n, e, message):
    cipher_text = []
    asc = Convert_Text(message) #convert string to ascii list
    for num in asc:
        cipher_text.append(FME(num,e,n))  #RSA to convert to cipher text

    return cipher_text

def Decode(n, d, cipher_text):
    message = ''
    decrypted = []
    for num in cipher_text:
        decrypted.append(FME(num,d,n)) #RSA decode to get ASCII in decrypted␣
 ↪list

    message = Convert_Num(decrypted) #convert ascii list to text
    return message

def factorize(n):
    for i in range(2, n-1):
        if n%i == 0:
            return i
    return FALSE
```

[3]:
```python
def display_menu():
    print("-----Hello! Welcome to the RSA Cryptography Menu------")
    print("Option 1: Encode a message")
    print("Option 2: Decode a message")
    print("Option 3: Break Key")
    print("Option 4: Get random primes to encode (up to 100)")
    print("Option 5: Exit Program")
```

[4]:
```python
import random
def main():
    while(True):
        display_menu()
        choice = int(input())
        #Choice 1 encode a message
        if choice == 1:
            #taking input
            print("Enter 1st prime number.")
            p = int(input())
            print("Enter 2nd prime number.")
```

```
            q = int(input())
            #calculating n and taking e input
            n = p*q
            print("Enter a key (e) value (65537 reccomended).")
            e = int(input())
            print("Enter your message")
            msg = input()
            encoded1 = Encode(n, e, msg) #encoding message
            print(f"Encoded Message: {encoded1}")
        #Choice 2 decoding a msg
        elif choice == 2:
            print("Enter the encoded message")
            encoded_msg2 = [int(num) for num in input()[1:-1].split(",")]␣
↪#split string by commas, ignore brackets and then convert each num to int

            #taking input
            print("Enter n value")
            n_2 = int(input())
            print("Enter private key d")
            d_2 = int(input())
            decrypted_2 = Decode(n_2, d_2, encoded_msg2) #decrypting message
            print(f"Decoded Message: {decrypted_2}")
        #choice 3 breaking a msg with no private key
        elif choice == 3:
            print("Enter encoded message")
            encoded_msg3 = [int(num) for num in input()[1:-1].split(",")]␣
↪#split string by commas, ignore brackets and then convert each num to int
            print("Enter n value")
            n_3 = int(input())
            print("Enter key (e) value")
            e_3 = int(input())

            #calculating prime numbers p and q
            p3 = factorize(n_3)
            q3 = n_3//p3

            #calculating private key d
            d3 = Find_Private_Key_d(e_3, p3, q3)
            decrypted_3 = Decode(n_3, d3, encoded_msg3) #decoding message
            print(f"Decoded Message: {decrypted_3}")
        #choice 4 generate prime nums
        elif choice == 4:
            primes = [2, 3, 5, 7,␣
↪11,13,17,19,23,29,31,37,41,43,47,53,59,61,67,71,73,79,83,89,97] #list of␣
↪prime numbers
            #picking random number in primes list
            prime1 = random.choice(primes)
```

```python
            prime2= random.choice(primes)

            if(prime2 == prime1): #condition to make sure p and q aren't equal
                prime2 = random.choice(primes )
            print(f"Prime 1: {prime1} Prime 2: {prime2}")
        #exit program choice
        elif choice == 5:
            print("Goodbye!")
            break
        #default case
        else:
            print("invalid option, try again")

main()
```

```
-----Hello! Welcome to the RSA Cryptography Menu------
Option 1: Encode a message
Option 2: Decode a message
Option 3: Break Key
Option 4: Get random primes to encode (up to 100)
Option 5: Exit Program

 5

Goodbye!
```

[ ]:

```python

```