RSA Encryption

Darin Critchlow
CSIS 2430-001

Objective:

Your assignment is to encrypt the following message "The Queen Can't Roll When Sand is in the Jar" using the values for $p = 71$, $q = 67$, and $e = 17$.

What Worked:

I was able to reuse the code that I had for modular exponentiation. I also used much of what I had already for the Caesar cipher.

What Didn't Work:

I had to implement my own way to split up the letter blocks

Comments:

I enjoyed walking through and studying the operations of RSA encryption. I was glad that I was able to use the function that I wrote to split up the letter blocks in more than one function in my code.

Code:

```
# ========================================================================#
#    Functions
# ========================================================================#

''' Give a letter a number value '''
def letterToNumber(message):
    numberedText = ''
    for character in message:
        if character.isalpha():
            if character.isupper():
                alphabetNumber = ord(character)-65
            if character.islower():
                alphabetNumber = ord(character)-97
            if alphabetNumber < 10:
                numberedText += '0'+str(alphabetNumber)
            else:
                numberedText += str(alphabetNumber)
        if character.isspace():
            numberedText += ' '
    return numberedText

''' Give a number a letter '''
def numberToLetter(message):
    letterText = ''
    letters = splitLetterBlocks(message,2)
    letter = letters.split()
    for l in letter:
        l = int(l)+65
        letterText += str(chr(l))
    return letterText

    '''
    Express GCD as a linear combination gcd(a,b) = sa + tb
    http://en.wikibooks.org/wiki/Algorithm_Implementation/
    Mathematics/Extended_Euclidean_algorithm
    '''
def extended_eculidean_gcd(a, b):
    x,y, u,v = 0,1, 1,0
    while a != 0:
        q, r = b//a, b%a
        m, n = x-u*q, y-v*q
        b,a, x,y, u,v = a,r, u,v, m,n
    return b,x,y
```

```
''' Private key exponent '''
def modular_inverse(a, m):
    g, x, y = extended_eculidean_gcd(a, m)
    if g != 1:
        return None  # modular inverse does not exist
    else:
        return x % m

''' Expand to base "b" '''
def base_expansion(n, b):
    q = n
    a = []
    while q != 0:
        a.append(q % b)
        q = q // b
    return a

''' Calculates Modular Exponentiation for base 2 '''
def modular_exponentiation(b, n, m):
    a = base_expansion(n, 2) # calculate for base 2
    x = 1
    power = int(b) % m
    for i in xrange(0, len (a)):
        if a[i] == 1:
            x = (x * power) % m
        power = (power * power) % m
    if len(str(x)) < 4:
        if len(str(x)) == 3:
            x = '0'+str(x)
        if len(str(x)) == 2:
            x = '00'+str(x)
        if len(str(x)) == 1:
            x = '000'+str(x)
    return x

''' Split message into given length '''
def splitLetterBlocks(letterNumbers, lengthToSplit):
    letterNumbers = letterNumbers.split()
    letterNumbers = ''.join(letterNumbers)
    numbers =''
    for i in range(0,len(letterNumbers),  lengthToSplit):
        numbers += letterNumbers[i:i+ lengthToSplit] + ' '
    return numbers

''' Encrypt message using RSA '''
def rsa_encryption(p, q, e, numbers):
```

```
    n = p*q
    totient = (p-1)*(q-1)
    number = numbers.split()
    encrypted = ''
    for num in number:
        encrypted += str(modular_exponentiation(num,e,n)) +' '
    return encrypted

''' Decrypt message using RSA '''
def rsa_decryption(p,q,e,numbers):
    n = p*q
    totient = (p-1)*(q-1)
    d = modular_inverse(e,totient)
    number = numbers.split()
    decrypted = ''
    for num in number:
        decrypted += str(modular_exponentiation(num,d,n)) +' '
    return decrypted

# ========================================================================#
#    Main
# ========================================================================#

if __name__ == '__main__':
    message = "The Queen Can't Roll When Sand is in the Jar"
    print '\nGiven message:', message
    messageNumber = letterToNumber(message)
    messageSplit = splitLetterBlocks(messageNumber, 4)
    encrypted = rsa_encryption(71, 67, 17, messageSplit)
    print '\nMessage encrypted:',encrypted
    decrypted = rsa_decryption(71, 67, 17, encrypted)
    print '\nMessage decrypted:', numberToLetter(decrypted)
```

```
[darin@darin-HP:~/Documents/CSIS2430Spring2014]$python rsa_encryption.py

Given message: The Queen Can't Roll When Sand is in the Jar

Message encrypted: 2406 1483 3994 1023 1875 3601 1428 3706 3352 1023 1477 0909 3225 4084 2406 4265 3048

Message decrypted: THEQUEENCANTROLLWHENSANDISINTHEJAR
```