DESIGN-ASGN6 “Public Key Cryptography”

Purpose:

The purpose of this assignment is to implement the principles of cryptography. The program will utilize the “Keygen” program to generate a public and private key. It will then use the “encrypt” program which will take in an input, encrypt it using the public key and store the encrypted message in a new text file. To decrypt the message, the “decrypt” file will take in the encrypted message and decrypt it with its private key and return the original message.

randstate:

```
randstate_init(seed: int):
     gmp.randinit(state)
     gmp.randseed(state, seed)

randstate_clear():
     gmp.randclear(state)
```

numtheory:

```
#cited from Professor Long’s Manual
power_mod(a, d, n):
     V = 1
     P = a
     While d >0:
          if(d%2 == 1):
               V = (v*p)%n
          p = p*p %n
          d = floor(d/2)
     Return v

#cited from Professor Long’s Manual
GCD(a, b):
     while(b!= 0):
          t=b
          b = a%b
```

```
        a = t
    return a

#cited from Professor Long's Manual
mod_inverse(a, n):
    r= n
    r_prime = a
    t = 0
    t = 1
    while r_prime != 0:
        q = floor(r/r_prime)

        r_temp = r
        r = r_prime
        r_prime = (r_temp - q)*r_prime

        t_temp = t
        t = t_temp
        t_prime = (t_temp -q)*t_prime

    if r >1:
        Return 0
    If t< 0
        t+=n

    Return t

# Cited from Professor Long's manual
def pow_mod(base, exponent, modulus):
    P = base
    V = 1

    While (exponent > 0):
        If (exponent%2 == 1):
            V = v*p % modulus
```

```
        P = p*p%modulus
        Exponent = floor(exponent/2)

# Cited from Profssor Long's manual
Def is_prime(n, iters):
    If (n%2 == 0 and n != 2):
        return false
    If (n == 2 or n == 3):
        Return true
    If (n < 2)
        Return false

    R, s = 0
    While(r % 2 == 0):
        s+=1
        r/=2
    for i in range(iters):
        Rand_val = random(2, n-2)
        Y = pow_mod(rand_val, r, n)
        if (y!=1 and y != n-1):
            J = 1
            While (j <= s-1 and y != n-1):
                Y = pow_mod(y, 2, n)
                If (y == 1):
                    Return false
                j+=1
            If (y != n-1):
                return false
    Return true

Def make_prime(bits, iters):
    base = sizeinbase(p, 2)
    while(!is_prime(p, iters) or base < bits):
        p=urandomb(state, bits)
        Base = pow(2, bits)
```

```
        p+=base
    Return p
```

RSA:

```
Def rsa_make_pub(p, q, n, e, nbits, iters):
    Nbit_low = nbits/4
    Nbit_high = (3*nbit)/4
    Rand_val_p = random(nbit_low, nbit_high);
    Rand_val_q = nbits - rand_val_p

    P = make_prime(p, rand_val_p, iters)
    Q = make_prime(q, rand_val_q, iters)

    Totient = (p-1)(q-1)

    E = urandomb(state, nbits)
    While(gcd(totient, e) != 0):
        E = urandomb(e, state, nbits)

def rsa_write_pub(n, e, s, username, pbfile):
  1) Writes out the inputs onto the specified pbfile using
     gmp_fprintf

def rsa_read_pub(n, e, s, username, pbfile):
  1) Reads out the pb file and assigns the variables their
     correct values from the file

def rsa_make_priv(d, e, p, q):
    Totient = (p-1)(q-1)
    D = mod_inverse(e, totient)

def rsa_write_priv(n, d, pvfile):
  1) Writes out the values n and d on the pv file

def rsa_read_priv(n, d, pvfile):
```

1) Reads out the values of n and d from the pv file

def rsa_encrypt(c, m, e, n):
     C = pow_mod(m, e, n)

def rsa_encrypt_file(infile, outfile, n, e):
1) Calculates the k value from the formula ((log_2(n)-1)/8)
2) Creates a buffer array and prepends the first elementt o 0xff
3) Reads the infile
4) Counts how many bytes there are in the infile stores all the bits in the buffer
5) Imports all the data from the buffer to m
6) Encrypts the message and then writes it out to the outfile

def rsa_decrypt(m, c, d, n):
     M = pow_mod(c, d, n)

def rsa_decrypt_file(infile, outfile, n, d):
7) Calculates the k value from the formula ((log_2(n)-1)/8)
8) Creates a buffer array
9) Reads the infile and stores the encryption in a variable called c
10) Decrypts c using rsa_decrypt
11) Exports the message to a count var
12) Writes the decrypted data to the outfile

```
Def rsa_sign(s, m, d, n):
     s=pow_mod(m, d, n)

Def rsa_verify(m, s, e, n):
     T = pow_mod(s, e, n)
     If (t == m):
          Return true
     Return false
```

Keygen:

1) Opts the user for specified inputs
1) This will open the files to write the public and private keys to
2) Sets the permission on the private key file
3) Initializes the random state
4) Makes the public and private key
5) Grabs the username environment and converts it to bits
6) Writes the public and private keys to their respective files

Encrypt:

1) Opts the user for specified inputs
2) Reads the public file containing the public key
3) Converts the username to bits and stores it in a variable that holds the username's bit information
4) Verifies the signature of the key
5) Encrypts the file and writes the encryption on a separate file

Decrypt:

1) Opts the user for specified inputs
2) Reads the private key file
3) Decrypts the inputted encrypted file with decrypt_file function