Design Document

Description

The purpose of these programs is to create a secure public key cryptography system using RSA encryption. This is asymmetric encryption, where the user's private key is used for decryption and the user's public key is used for anyone to encrypt messages. The two most important functions here are encryption of a message: $E(M) = M^e (mod\, n) = C$, and decryption of a message: $D(C) = C^d (mod\, n) = M$. It is important to note that these functions are inverses, so $E(D(C)) = C$ and $D(E(M)) = M$. Finally, the third program is the keygen program which will be used to generate private and public keys.

The most challenging part of this assignment is that it requires numbers of around 256 bits. The unsigned integer in C can only hold 64 bits. To solve this, we use the <gmp.h> library and use mpz_t for the mathematical operations.

Structure

This section contains the pseudocode for the keygen, encryptor, and decryptor.

Keygen

```
func public_key_gen(int n){
p, q = two large primes with ≥ n bits // using Miller-Rabin function
phi = (p-1) * (q-1)
e = random number with ~ n bits
while(gcd(e, phi) > 1){
      e = random number with ~ n bits
}
return p*q, e
}
func private_key_gen(int p, q, e){
      phi = (p-1) * (q-1)
      d = e (mod phi)
      d = 1 / d
      return d
}
u = username
signed = u^d (mod n)
n, e = public_key_gen()
d = private_key_gen()
write_file(n, e)
write_file(d)
print(username, signed, p, q, n, e, d)
```

Encryptor

```
n, e = read(public_key)
if(verbose) print(username, signed, n, e)
s = signature
t = s^e (mod n)
if(t != username){
      return "Error: Invalid signature."
}
rsa_encrypt_file(message)

func rsa_encrypt_file(m) {
      k = [log2(n) - 1] / 8
      int block[k]
      block[0] = '0xFF'
      int read = read k-1 bytes from infile
      while(read > 0) {
            block = number of read bytes
            encrypted = block^e (mod n)
            write_file(encrypted)
      }
}
```

Decryptor

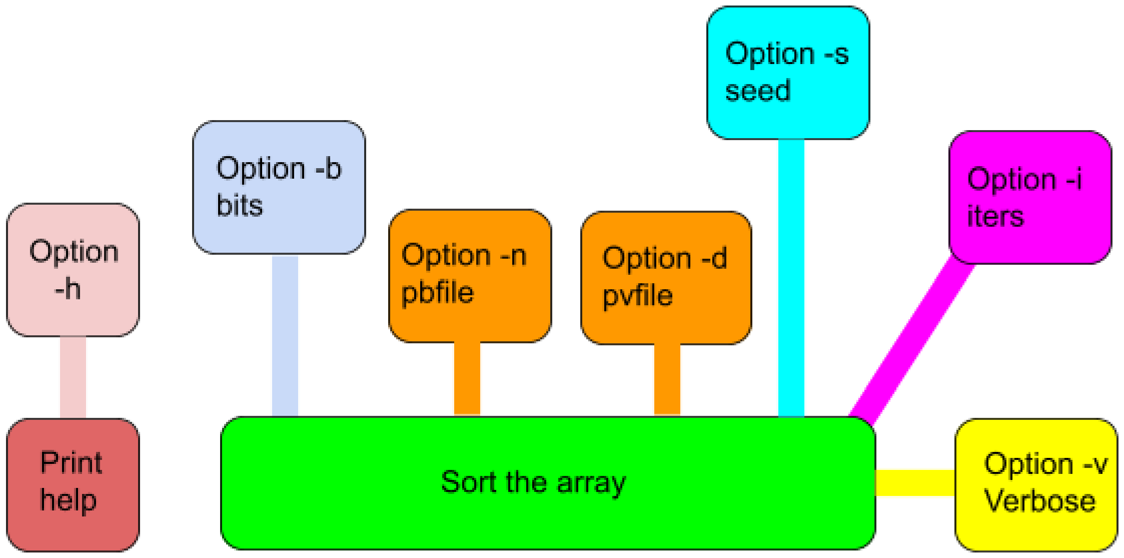
```
d = read(private_key)
if(verbose) print(n, e)
rsa_decrypt_file(message)

func rsa_decrypt_file(m) {
      k = [log2(n) - 1] / 8
      int block[k]
      int read = read read k-1 bytes from infile
      while(read > 0) {
            block = number of read bytes
            decrypted = block^d (mod n)
            write_file(decrypted)
      }
}
```

Inputs and Outputs

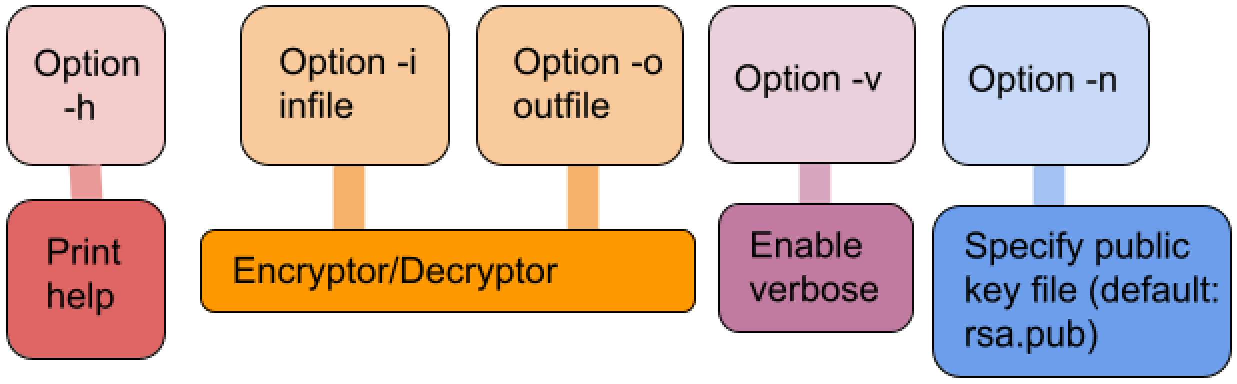
Key Generator expected inputs and outputs:

- The default number of bits is 256.
- The default number of iterations is 50.
- The default public key file is rsa.pub.
- The default private key file is rsa.priv.
- The default seed is set to time(NULL).



Encryptor/Decryptor expected inputs and outputs:

- The default infile is stdin.
- The default outfile is stdout.
- The default public/private key file is rsa.pub/rsa.priv.



Option -h
Option -i infile
Option -o outfile
Option -v
Option -n
Print help
Encryptor/Decryptor
Enable verbose
Specify public key file (default: rsa.pub)