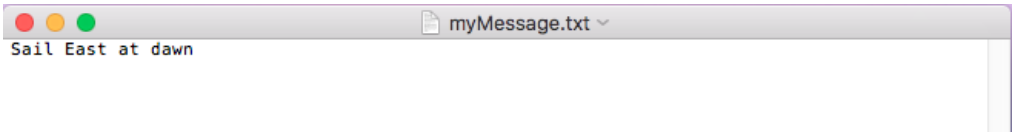
messageEncrypt

I. Encrypt

The program prompts the user to select encrypt or decrypt. If the user selects encrypt, the program will prompt the user to input the name of the text file containing the message to be encrypted. It will then output a text file containing an encrypted message, a text file containing a public key, and a text file containing a private key.

If we have a .txt file myMessage:



saved in the current directory, and we run:

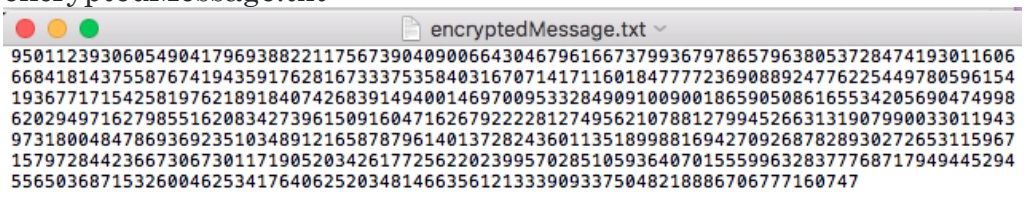
```
Ryans-MacBook-Pro:message-encrypt ryantine$ python messageEncrypt.py
```

we will receive following prompts and answer them so to encrypt myMessage.txt:

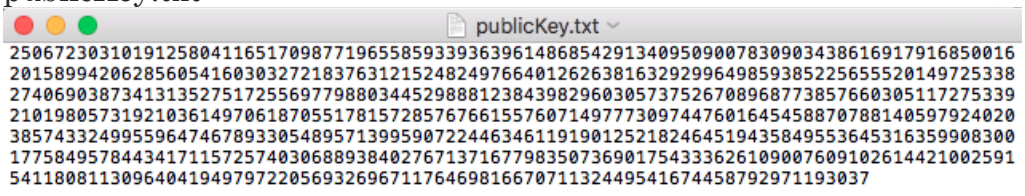
```
Would you like to encrypt or decrypt a message? encrypt
Enter the name of the text file containing your message to be encrypted or decrypted: myMessage.txt
Please find a file named encryptedMessage.txt containing your encrypted message, a file named publicKey.txt
containing your public key which may be stored anywhere and a file named privateKey.txt which must be stored
 safely
```

The following files will then appear in the current directory:

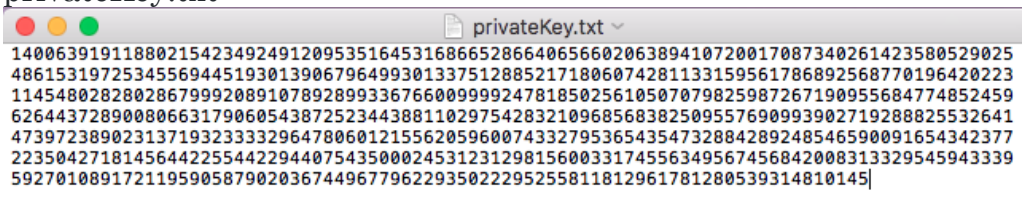
encryptedMessage.txt:



publicKey.txt:



privateKey.txt:



II. Decrypt

If the user selects decrypt, the program will then prompt the user to input the name of a text file containing an encrypted message, the name a text file containing the public key associated with that message, and the name of the text file containing the private key associated with that message. The program will output a text file containing the decrypted message.

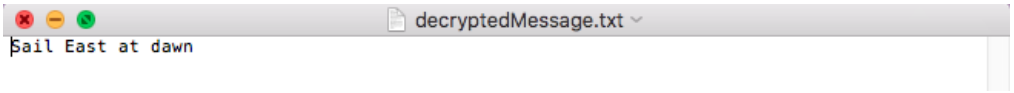
If we have the text files containing an encrypted message, and the public and private keys associated with that message saved in the current directory, and we run:

```
Ryans-MacBook-Pro:message-encrypt ryantine$ python messageEncrypt.py
```

we will receive the following prompts, and answer so to decrypt our message (we will continue from our example above and use the files encryptedMessage.txt, publicKey.txt, and privateKey.txt; these could be saved under any names when executed by the user).

```
Would you like to encrypt or decrypt a message? decrypt
Enter the name of the text file containing your message to be encrypted or decrypted: encryptedMessage.txt
Enter the name of a the text file containing the public key: publicKey.txt
Enter the name of the text file containing private key: privateKey.txt
Please find a file named decryptedMessage.txt containing your decrypted mesaage
```

The following file will appear in the current directory:



Which contains our decrypted message!

III. Implementation

This process is completed using optimal asymmetric encryption padding and the RSA algorithm. We imported SHA.256 from Crypto.Hash for hash functions when padding. In the RSA algorithm, the Rabin Miller primality test is used to determine if very large random numbers are prime:

```
def isPrime(num):
    """Return True if the number is prime, and False otherwise."""
    # Two basic cases where number can be quickly seen as prime or not prime
    if (num<2):
        return False
    elif num%2==0:
        return num == 2 # Two is the only even prime number

    # Start of Rabin Miller Primality Test
    s = num - 1
    counter1 = 0
    # if s is even keep halving it
    while s%2 == 0:
        s = s/2
        # increment counter1
        counter1 += 1
    trial = 8 # Number of times we will check num's primailty. Accuracy is improved with increased trials
    while trial>0:
        rando = random.randrange(2,num)
        modNum = pow(rando,s,num) # modNum is equal to rando^r mod num
        if modNum != 1: # Rabin miller test does not apply if v = 1
            counter2 = 0
            while modNum != (num - 1):
                # case that would mean that num is not prime
                if counter2 == counter1-1:
                    return False
                else:
                    # increment counter2
                    counter2 += 1
                    # update v
                    modNum = (modNum**2)%num
        trial -= 1
    # if none of these conditions have been met, num is likely tru
    return True
```

Function generateLargePrime() calls isPrime() which implemnts the rabin miller test:

```
def generateLargePrime():
    """Returns a prime number in the range from 2^1023 to (2^1024)-1"""
    while True:
        num = random.randrange(2**(1023), 2**(1024)-1)
        if isPrime(num):
            return num
```

The encryption algorithm also features a recursive implementation of the extended GCD algorithm, which is used to calculate modular inverse:

```
def extendedGCD(a, b):
    """Returns gcd, x and y so that a*x+b*y = gcd(x,y)"""
    # Base case (when a = 0)
    if a == 0:
        return (b,0,1)
    # Recursive case
    else:
        gcd, x, y = extendedGCD(b%a,a)
        return (gcd, y-(b/a)*x, x)

def modularInverse(a,mod):
    """Returns the value whose product with (a % mod) is equal to 1"""
    gcd,x,y = extendedGCD(a,mod)
    return x%mod
```