I have recorded my initial code at this development stage of the assignment below. There are still edits to be made of course:

```python
import numpy as np
from sympy import mod_inverse, isprime
import secrets
import random
from math import gcd

def is_probably_prime(x):
p = isprime(x)
return p

def bits(num):
return secrets.randbits(num // 8 + 1)

def attempts(number):
return secrets.randbelow(number)

def generate_large_primes(bits, attempts):
for _ in range(attempts):
candidate = secrets.randbits(bits)
if is_probably_prime(candidate):
return candidate
raise ValueError(f"Failed to generate a prime number of {bits} bits in {attempts} attempts.")

def power_mod(base, exponent, modulus):
result = 1
b = base % modulus
while exponent > 0:
if (exponent % 2) == 1:
result = (result * b) % modulus
exponent = exponent >> 1
b = (b * b) % modulus
return result

def euler_totient(n):
#Calculate Euler's Totient Function φ(n)
if n < 1:
return f"Invalid input: n must be a positive integer."
value = n
while p * p <= n:
if n % p == 0:
while n % p == 0:
n //= p
```

```
value -= value // p
p += 1
if n > 1:
value -= value // n
return value
def e(phi):
e = 65537 # Standard RSA public exponent
while gcd(e, phi) != 1:
e += 2
return e

def private_key(e, phi):
d = mod_inverse(e, phi)
if gcd(e, phi) != 1:
raise ValueError("e and phi are not coprime; modular inverse does not exist.")
else:
return d
def n(p, nq):
return np * nq

def encrypt(message, e, n):
cipher = power_mod(message, e, n)
return cipher

def decrypt(ciphertext, d, n):
message = power_mod(ciphertext, d, n)
return message

message = input("Enter a message: ")
cipher = encrypt(message, e, n)
decrypted_message = decrypt(cipher, d, n)
print(f"Ciphertext: {cipher}")
print(f"Decrypted message: {decrypted_message}")
```