```
import secrets
import math
from sympy import isprime, mod_inverse
from math import gcd

def generate_large_prime(bits):
while True:
candidate = secrets.randbits(bits)
if isprime(candidate):
return candidate

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

def euler_totient(n):
result = n
p = 2
temp = n
while p * p <= temp:
if temp % p == 0:
while temp % p == 0:
temp //= p
result -= result // p
p += 1
if temp > 1:
result -= result // temp
return result

def choose_e(phi):
e = 65537
if gcd(e, phi) == 1:
return e
e = 3
while gcd(e, phi) != 1:
e += 2
return e

def string_to_int(s):
return int.from_bytes(s.encode(), byteorder='little')
```

```
def int_to_string(i):
length = math.ceil(i.bit_length() / 8)
return i.to_bytes(length, byteorder='little').decode()

# ---------- RSA SETUP ----------

bits = 32
p = generate_large_prime(bits)
q = generate_large_prime(bits)

n = p * q
phi = euler_totient(n)

e = choose_e(phi)
d = mod_inverse(e, phi)

# ---------- ENCRYPTION ----------

message_str = input("Enter a message: ")
message = string_to_int(message_str)

if message >= n:
raise ValueError("Message too large for RSA modulus. Use a shorter message.")

cipher = power_mod(message, e, n)

# ---------- DECRYPTION ----------

decrypted_int = power_mod(cipher, d, n)
decrypted = int_to_string(decrypted_int)

# ---------- OUTPUT ----------

print("p =", p)
print("q =", q)
print("n =", n)
print("phi =", phi)
print("e =", e)
print("d =", d)
print("Ciphertext:", cipher)
print("Decrypted:", decrypted)
```