# Cryptography Homework 4

## Required Reading

Cryptology 4 slides

## Computing Keys

Note: The functions GCD() and inverse() are imported from PyCryptodome, which must be installed for this to work.  
from Crypto.Util.number import GCD, inverse

This lab allows us to play with the math behind RSA encryption. It is “schoolbook RSA”, and is **\*not\*** safe to use for encrypting data outside of teaching labs and CTFs. Remember that real RSA protects itself against attacks using PKCS#1 hashing and padding.

To gain more practice in Python, we will compute exponents (d, in the slides) for a private key modulo Φ, and and again for modulo n. The first one should work, and the last one using modulo n should not (if it did, RSA would be broken. The fact that it fails when we use modulo n demonstrates that you must have p or q to compute d.) We will use the procedure below for both attempts. Fill in your results in the form in the Turn In section.

### Alice Generates a Key

1. Use prime numbers p = 131 and q = 157
2. Compute n = p \* q
3. Compute either
   1. phi = (p – 1)(q – 1))
4. Pick a small number for e, for the public key
5. Compute GCD(e, phi). The GCD must equal 1, which means e and phi are relatively prime.
6. Compute the number, d, for the private key from d = inverse(e, phi).
7. Alice gives public key [n, e] to Bob, keeps private key [n, d] secret

### Bob Encrypts a Message

1. Pick an integer, plaintext, that is less than n, that will represent Bob’s message. We will pretend that it decodes into a few ASCII letters.
2. Bob encrypts m with Alice’s public key using ciphertext = pow(plaintext, e, n)
3. Bob gives ciphertext to Alice

### Alice Decrypts the Message

1. Alice computes plaintext = pow(ciphertext, d, n)
2. The answer should be the same as the plaintext Bob started with

## Repeat with n, doomed to failure

Let us assume that an attacker has n and e from the public key, but they do not have p or q so they cannot compute Φ(n). Since they are wishful thinkers, they try to compute the private exponent d using inverse(e, n). This should fail miserably. If it succeeds, it will mean that RSA encryption does not work.

What does failure look like? If you encrypt a message and then try to decrypt it using an incorrectly computed d, the decrypted message will be entirely different from the original message.

# Turn in

This data is the same for both attempts:

p = 131

q = 157

n = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ p\*q

### Attempt 1, using Φ (this should work -> decrypted = original plaintext)

phi = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (p-1)\*(q-1)

e = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ pick this yourself

gcd(e, phi) = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ must equal 1. If not, pick a new e

d = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ inverse(e, phi)

public key [n, e] \_\_\_\_\_\_\_\_\_\_\_\_\_\_

private key [n, d] \_\_\_\_\_\_\_\_\_\_\_\_\_\_

plaintext integer \_\_\_\_\_\_\_\_\_\_\_\_\_\_ pick this yourself, < n

ciphertext integer \_\_\_\_\_\_\_\_\_\_\_\_\_\_ pow(plaintext, e, n)

decrypted integer \_\_\_\_\_\_\_\_\_\_\_\_\_\_ pow(ciphertext, d, n)

### Attempt 2, using n (this should fail—decrypted not equal to plaintext)

n = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ n, same as above

e = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Use the same e that you did in Attempt 1

d = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ inverse(e, n)

public key [n, e] \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

private key [n, d] \_\_\_\_\_\_\_\_\_\_\_\_\_\_

plaintext integer \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Use the same plaintext that you used in Attempt 1

ciphertext integer \_\_\_\_\_\_\_\_\_\_\_\_\_\_ pow(plaintext, e, n)

decrypted integer \_\_\_\_\_\_\_\_\_\_\_\_\_\_ pow(ciphertext, d, n)