# Cryptography Homework 4

## Required Reading

Cryptology 4 slides

## Encrypting with RSA

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. Real RSA protects itself against attacks using hashing and padding (PKCS#1).

To gain more practice in Python, we will compute exponents (d, in the slides) for a private key modulo, Φ, and n. Fill in your results on the form in the Turn In section.

### Alice Generates a Key

Note: I use the Greek letter Φ below, but you cannot use those letters in your Python code. Also, “lambda” is a reserved word in Python, so you should not use that either. (I just like Greek letters.) Replace Φ with your own variable names.

1. Use prime numbers p = 131 and q = 157
2. Compute n = p \* q
3. Compute Φ = (p – 1)(q – 1))
4. Pick a small number for e, for the public key
5. Compute GCD(e, Φ). The GCD must equal 1, which means e and Φ are relatively prime.
6. Compute the number, d, for the private key from d = inverse(e, Φ).
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

## Hand In #1

p = 131

q = 157

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

Φ(n) = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (p-1)\*(q-1)

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

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

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

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)

## An RSA Decryption Puzzle

This is a problem from the spring 2022 NCL Individual game.

Graphical user interface, text, application, email

Description automatically generated

n = 591

e = 157

c = 290 279 275 141 247 290 557 374 141 582 255 227 291

There is a problem: n is too small!

## Hand In #2

What are the factors of 591? Hint: Google can be your friend.

Follow the procedure in Cryptology4-Public-Key-Intro-RSA, slides 10 & 12, to recreate Alice’s public key. For n, use the value in Alice’s public key, above. For p and q, use the factors of n that you just found. Remember to use  
from Crypto.Util.number import inverse

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

e and d are multiplicative inverses, modulo phi

So, d = inverse(e, phi)

## Hand in #3

What is d?

Now that you know n, p, q, d, and e, and n, you can decrypt the message.

Slow way: You can use the Python pow function as in slides 9 & 13 to decrypt each number. For the first number, it would be  
pow(290, d, 591)  
where you would insert your value of d.

Then you could look up each number in an ASCII table. Or, you could have Python do the ASCII lookup for you with the chr function.  
chr(pow(290, d, 591))

Or you could write a short Python script where you put c (the ciphertext) into a list and loop through it.

## Hand In #4

What is the flag?