CS 378

RSA Project Report

Cameron Lydon

My program runs on a single call “python RSA.py”. I imported three different python modules; random, for generating random integers, math, for the built in greatest common divisor function and ‘sys,’ for calling a system exit. The main three ‘modules’ of my program are Key Setup, Encryption and Decryption. In order for everything to run smoothly, message.txt must only have one integer.

For setting up the public and private keys, I implemented a few different algorithms. The first code I wrote isn’t an algorithm, but it generates a random integer p, from $2^{1024}\ to\ 2^{2048}$. And then it generates a second number from $2^{1024} - 1\ to\ p - 1$. Next, I wrote code for the Fermat Primality Test which takes in a list of 2 integers, generated from the previous function. When I was writing this code, I found out about a function native to python called pow( ) usually used for raising a number to a power, but when given three parameters, it performs modular exponentiation. My function called ‘Fermat’ returns the number it is testing if it is prime, otherwise it will return 0. This is important for later. Next, I implemented the Extended Euclid Algorithm to find modular inverses.

When it came time to run everything, I set up my project so that it would generate two large primes, p and q. Then, I computed $n = p * q$ and $\phi(n) = (p - 1)(q - 1)$. To generate e, I first set it to $2^{16} + 1$ and then ran a while loop to make sure the greatest common divisor between e and $\phi(n)$ was one, increasing e by one each time it was not. After running the program a few times, I can confirm that $2^{16} + 1$ usually works. Once e was set, I used the Extended Euclid Algorithm to compute $d\ \equiv e^{-1}\ mod\ \phi(n)$. Then I wrote n and e to public_key.txt and d to private_key.txt and then I was ready to encrypt and decrypt.

For encryption I first opened public_key.txt and since I needed to grab both integers I used ‘with open ( )’. However, when I open message.txt I use readline( ). I couldn’t figure out how to use readline with public_key.txt because there were two integers in it. The next issue I ran into was to cast the strings I read in from the file as integers. I quickly overcame this by using int( ). Using pow( ) again I computed the encryption as $c \equiv m^e\ mod\ n$. I then saved the ciphertext (c) in ciphertext.txt.

Since I had already figured out a way to read from a file with a single integer and a file with two integers, running the decryption was straightforward. I read inputs from public_key.txt and private_key.txt and computed the decryption as $m \equiv c^d\ mod\ n$. I saved the decrypted message (m) in decrypted_message.txt.