Testing RSA Encryption and Decryption

Moore Macauley
University of California, Santa Cruz

November 4, 2022

1 Introduction

For this assignment, I was instructed to reimplement a key generator, an encryptor, and a decryptor for use with RSA encryption. I was also tasked with testing the functions I had written, in order to ensure their functionality. To do this, I began by rigorously testing the smallest, simplest functions that others relied on, and moving my way up to functions that depended on those simpler functions

2 Number Theory

For these functions, my method of testing them was fairly simple. While we were not allowed to use gmp number theory functions in our code, we could test against them. So for the functions gcd, pow_mod, mod_inverse, and is_prime, I created a for loop that would be iterated over 100,000 times, generated random numbers using gmp_urandomb, before using the corresponding gmp function (mpz_gcd, mpz_powm, mpz_invert, and mpz_probab_prime_p respectively) on those random numbers. If this result was not equal to what was generated with my own function, I would print out the numbers used, and the respective results.

When using this method, I started with relatively small random numbers, usually around 32 or 64 bits long, so I could use a calculator to try and determine what had gone wrong. I would then run the test again with larger random numbers, usually 1000 bits long, as these would be closer to values these functions could expect to use. Once both situations worked, I was confident that my number theory function worked.

Make prime was a little bit different. To begin with, I used a shorter loop, as make prime has a long runtime. I then took the value generated by make prime, and used mpz_probab_prime_p on it, to see if it actually was prime. If the number produced was not prime, I printed out what "prime" number it failed on. Once I had no failures, I was confident that make prime was functional.

3 RSA functions

For these functions, I started by writing all of the key generating functions, that is, rsa make and write public and private keys, alongside rsa sign which is required for making a public key. Then, using the keys generated by these functions, I use encrypt-dist and decrypt-dist to encrypt and then decrypt a file. As I know encrypt-dist and decrypt-dist work so long as the keys are valid, if they can successfully encrypt and decrypt with my keys, I know they keys must be valid. As such, I can be confident that my key generating functions are functional.

I then move onto the encryption and decryption functions and do much the same thing. I encrypt a file with the encryption functions I wrote, using keys from keygen-dist, before decrypting with decrypt-dist. This will only work properly if my encrypt function is encrypting properly, so I know it must work. I then apply the same method to my decrypt function with keygen-dist and encrypt-dist to determine its functionality.

4 Main Functions

Finally, I had to test the main functions. While in theory all of the rsa functions should work in keygen, encrypt, and decrypt, just to be safe I do the RSA tests again. I then test a variety of command line inputs to the -dist programs, and see how they react. Based on this, I then make the keygen, encrypt, and decrypt function the same way.

5 Functionality

While it has yet to happen to me on the gitlab test machine, on my own computer, my keygen program will occasionally take longer than 20 seconds when generating keys of 4092. My own computer does seem to be considerably slower than the gitlab computer, with keygen-dist regularly taking 2 seconds, which is slower than the piazza post containing keygen-dist runtimes. As such, it is possible that it is simply my computer being slow that is making my keygen take longer than 20 seconds, but I figured I'd mention it here, just in case.