Assignment 5 WRITEUP.pdf

Julian Chop

March 1, 2023

1 Public Key Cryptography

Cryptography, which was once used by governments, spies, and the military, is now something everyone uses in their day-to-day, especially because of how much technology is ingrained into daily life. Cryptography is used for security. To make sure information is only accessible to the intended people and places. In this assignment, we looked at different ways information is encrypted/decrypted through different algorithms.

The Schmidt-Samoa (SS) algorithm is a public-key cryptography algorithm and was the focus of this project. The SS algorithm is a modified version of the RSA algorithm, another public-key cryptography algorithm most commonly used. I'll quickly explain the difference between the two.

Tee way RSA encryption works is that it relies on the difficulty of factoring the product of very large prime numbers. One way someone would decrypt RSA ciphertexts could be to factor all ciphertexts encrypted under that key, which would take a while(like a very long time). This is known as the RSA problem. Though with advancements in technology and computers, this method of cracking RSA is not impossible. This is where SS comes in. SS modifies the RSA algorithm and makes it so the RSA problem is more similar to the factoring problem. The reason why RSA is still more commonly used though is that it's more efficient.

2 Use in daily life? In the world?

As said earlier, since we are in a time where there is so much data and information being passed around through our devices, cryptography is a necessity if we want our information to be secure. We only want this information to be accessed by those authorized. Whenever you send an email, send a text message, make an online purchase, or access your bank, cryptography is used. Basically, cryptography ensures that the correct person is receiving your data.

3 Understanding My Code

Overall, I think the concepts of this assignment were pretty easy to understand. We were tasked with creating a key-pair generator, an encryptor that used the generated key pair to encrypt a given file, and a decryptor that used the corresponding private key to decrypt the file.

First, the public key is generated by the product of two large, random prime numbers, p, and q. The public key is computed as p * p * q. You can see where utilizing the factoring problem comes in. Once we get our public key, we can make its corresponding private key. This is found by finding the inverse of the least common multiple of p-1 and q-1.

Once you have the public and private key pairs, it's a matter of making the encrypt and decrypt programs. For encrypt, you take in a file and read it in blocks of data. For every block, encrypt it using the public key and print it to the given output. Then for decrypt, you take in a file encrypted by the public key pair and read it in the same size blocks as the way it was encrypted. Then decrypt the file block by block until the entire file is decrypted.

4 Challenges and Mishaps

This assignment was hard. I don't think it was the understanding of what to do that was difficult, it was actually implementing what we had to do that was. That and some just some stupid mistakes that took like a line of code to fix.

One issue I had was with generating a public key that had to be at least a certain amount of bits. Sometimes, my program would work, generating a public key with the required amount of bits, or a little more. Other times, it would go 1-2 under the required which is a no no. I spent hours trying to figure out my issue. The issue ended up being how bit addition worked. Basically, the two numbers that would multiply together to make the public key, even though they would have the required bit length, would not make a number with the required bit length. So what I ended up doing is after I generated a random number with make_prime, I just set the required bit length to 1.

Another problem I had was with taking data from a file to encrypt it. What I had initially done was read the file in blocks and then print out the whole block to the output. This would work when there was enough data to fill the allocated block. But once the amount of data was less than the block, it would print out the data, and whatever was left over from the previous read. To solve this, I just printed out only the read data instead of the whole block.

5 We Learning!

Aside from learning about cryptography, this assignment got me some more C programming. At this point of the quarter, I can say I am pretty familiar with C and this assignment proved it. Though it did take a while, I ended up finishing it! I also got some experience with using large variables. I'd say the GNU multiple precision arithmetic library was one of the harder things in this assignment. Instead of just adding two variables together, you would have to use a mpz_add(x,y,z). It would make taking simple ideas harder to implement. But by the end of the assignment, I got pretty used to it.

6 Citations/Resources used

- I went to Tuesday's section from 7:00-8:30 with tutor John in the first week of the assignment. He get us started by explaining how we might approach some of the functions in numtheory.c and how to use mpz_t variables/arithmetic.
- For the functions in numtheory.c, I followed pseudo-code from resources.
- I also used a formula to get a random number within a certain range for my ss_make_pub() function. The equation is: num = (rand_num % (upper-lower+1)) + lower. Here is where I got the equation. `https://www.geeksforgeeks.org/generating-random-number-range-c/`
- The Python folder in resources that has some examples of making and checking prime numbers was also a big help.

- I also went to the GMP Documentation for understanding certain gmp/mpz functions and how to use them. Here's the link to it: `https://gmplib.org/manual/Concept-Index`