Encryption: a Key Aspect of Electronic Communication

Isabella Phung

February 2023

1 Introduction

This documents the approximate development and learning process for this library which implements Schmidt-Samoa encryption. The DESIGN.pdf documents focuses on the functions and their pseudocode. The README.md file describes how to compile and run this library.

2 Small Errors

When using the scan-build make, a majority of the errors it brought up were warnings for printf statements that the inputfilename for encryptor and decryptor was not initialized. This variable holds the string for the input file if the user chooses to input a file name of their choosing, this value is not initialized as the print statement is hidden in an if statement that will only run if the value is initialized already, so there's no possibility of the value being printed prior to initializing.

The only other error was that, when allocating my blocks, I wrote: uint8_t *block = (uint8_t *) calloc(blocksize, sizeof(char)). It states that uint8_t is incompatible with sizeof operand type 'char'. Chars and unsigned eight bit integers are equivalent, so the file runs without issue. I could have instead declared something like char *block = (char *) calloc(blocksize, sizeof(char)) instead, but the context that this is used for, where I have to go back and forth between text and bytes, it provides more clarity to me that this isn't just a string, but an array of both numbers and characters that are going to be encrypted or decrypted.

3 Overall Concepts

There are some projects in this class which I have dabbled with before mainly in different classes with different programming languages. For example, Conway's Game of Life or our sorting project, but encryption is a subject I haven't touched upon yet at all. I can't say that this subject is one I fully understand inside and out (well, there aren't many subjects I am that fluent in admittedly). I still stare at the power mod or the mod inverse algorithm and squint in confusion because I don't genuinely understand exactly why one value has to be another, but after implementing it and testing it over and over again, I can safely say that the math of it works. Initially, reading over the algorithms made me feel intimidated at my lack of mathematics fluency, but the example python code was extremely helpful in explaining exactly what they meant. For example in the Miller-Rabin primality algorithm, I was completely lost when it listed "write n-1 = 2^sr such that r is odd", but it just covers some of the early cases, such as 0, 1 and any even number that isn't 2 are all not prime numbers, but 2 and 3 should return "prime".

Before encryption was just kind of a vague amalgamous concept in my brain that required some magic math to make happen, but I now know some of the key aspects that contribute to modern encryption: prime numbers, power mods, and inverse mods, as well as the idea that factoring large numbers is difficult; these all are foundational to most of our well known encryption algorithms.

One other big concept that stuck with me was that it's not the algorithm that needs to be kept safe but rather the keys. I conceptualized this as something close to the basic ideas of real life keys. The way keys by themselves work isn't kept secret (a cylinder with pins that shift up and down), anyone can also buy a lock for themselves and learn how to install it. But just because someone knows how a lock works doesn't mean they can open every door.

Although I don't necessarily understand every single aspect of the mathematics, I did find one aspect particularly interesting in regards to keys was that having many parts helps with security. Though having multiple parts can cause issues in the event one piece of the puzzle were to be lost or confusion in how the pieces fit together, if it's a computer that's handling the math necessary to make the key work, then it makes far a fairly reliable system. Instead of having one singular key, using two keys, each with multiple important numbers, it becomes much harder to crack or figure out via brute force.

4 Notable Struggles

There was certainly a fair few segmentation faults I got while working on this project, probably the most annoying factor was dealing with GNU Multi-Precision. I had spent ages trying to figure out why I was getting a segfault only to find out I needed a NULL whenever I use mpz_clears or mpz_inits. But I can certainly see the benefits how GMP allows for numbers of effectively infinite size. It makes some aspects of the code obscure since we no longer have symbols that represent operations and we can't perform certain operations in the same line, but the speed makes GMP an appealing library.

5 Resources

How to print part of a string
How to use chmod in c
Linker "Unused Package Error"
How to use pkg-config in makefiles
Additional Reading on External Variables
Undefined Global Variable Error
Additional GMP Examples
Offical GMP Documentation
Don't forget the NULL when initializing or clearing multiple GMP variables