# Assignment 5 Design

Caitlin Smith

February 20, 2023

## Program Description

In this assignment, a message is able to be encrypted and then decrypted. Public and private keys used for the encryption and decryption are generated using the Schmidt-Samoa (SS) Algorithm. All of the math and functions for the program are within the numtheory.c, randstate.c, and ss.c files. There are separate files for key generation, encryption, and decryption. Those three files utilize command-line arguments in the main function in order to set specifications for file access.

## 1 randstate.c

**Purpose:** This file contains the random state functions used in the other files. The interface is given in the randstate.h file.

**Pseudocode:**

Include standard .h files and randstate.h.

Initialize the random state variable.

Create a void function for randstate initialization with parameter seed.
Call srandom() passing the seed.
Call gmp randinit passing state.
Call gmp unsigned randseed passing state and seed.

Create a void function that clears and frees memory.
Call gmp randclear, passing state.

# 2 numtheory.c

**Purpose:** This contains the mathematical functions used for the SS library functions. The interface is given in the numtheory.h file.
**Pseudocode:**

Include the necessary standard and the numtheory.h file.

Create a void function for modular exponentiation that takes parameters out, base, exponent, and modulus). It stores the result in the out variable.
Set temp variables so that the passed variables are not the ones being changed, except for out.
Set variables v to 1, p to base, and d to exponent.
While d is greater than 0,
    If d is odd,
        Set v to the product and v and p mod the modulus parameter.
    Set p to the product of p and p mod the modulus parameter.
    Set d to d divided by 2, using floor division.
Set the out variable to v.

Create a function that Utilizes Miller-Rabin testing to check if a number is prime. It returns a boolean and takes in two parameters n and iters.
Check if n is less than 0, 2, 3, or even. These are special cases that will return, false, true, true, and false, respectively.
Create variables for s and r, setting r to n - 1.
Create a while loop that will run until r is odd.
    Divide r by 2 using floor division.
    Increment s by 1.
Create a for loop that will run for the iterations passed as the iters parameter, starting at 1.
    Set a variable a to be a random number in the range (2, n-2).
    Set a variable y to the result of pow_mod(a, r, n).
    If y isn't equal to 1 and it isn't equal to n - 1,
        Set a variable j to 1.
        Create a while loop to run until j is greater than s - 1 and y is equal to n - 1.
            Set y to the result of pow_mod(y, 2, n).
            If y is equal to 1, return false.
            Increment j by 1.
        If y is not equal to n - 1, return false.
True is returned if the loop is completed and it hasn't entered an if state that returned false.

Create a void function void that will make a random prime number, passing p, bits, and iters.
Create a flag for the while loop and set it to false.
Create a while loop that will run until a prime of the right conditions is found (the flag is true).
    Set p to a random number.

If p is greater than or equal to the number of bits desired,
If p is prime (call the is_prime function), stop the loop by setting the flag to true.

Create a void function to compute the greatest common divisor of the parameters a and b. The result is stored to d.
Set temp variables equal to a and b, so those are not changed.
While temp b is not equal to 0.
Set temp t to the value of temp b.
Set temp b to the value of temp a mod temp b.
Set temp a to the value of temp t.
Set d to temp t.

Create a void function that will compute the inverse of the parameters a mod n. It takes in parameters i, a, and n.
Create variables for r, r', t, and t'. Set them to n, a, 0, and 1, respectively.
While r' is not equal to 0.
Create temp variables for r, r', t, and t'. Set then all to the values in the original variables.
Create temp variables for q and the two multiplications done later.
Set q to r divided by r'.
Set r to the temp r'.
Set the multiplication var to q times temp r'.
Set r' to the temp r minus what was just multiplied.
Set t to the temp r'.
Set the other multiplication var to q times temp t'.
Set t' to temp t minus what was just multiplied.
Clear all of the temps from this loop.
If r is greater than 1, set t to 0.
If t is less than 0, set t to t plus n.
Set i to t.
Clear r, r', t, and t'.

# 3 ss.c

**Purpose:** This file contains the library of SS functions needed for reading, writing, encrypting, and decrypting. The interface is given in the ss.h file.
**Pseudocode:**

Include the necessary standard and the ss.h file.

Create a void function to create parts of a new public key (p, q, and n) It takes in parameters p, q, n, nbits, and iters.

Set the number of bits for p to a random number in the correct range.
Set the number of bits for q to nbits minus two times pbits.
Create a while loop that will run until the correct primes are found (within the correct bit range and the right non-factor specifications).
    Call make_prime with parameters p, pbits, and iters.
    Call make_prime with parameters q, qbits, and iters.

Create a void function to write a public key to a file. It takes parameters n, username, and pbfile.
Print n and username to the pbfile using hex strings.

Create a void function to read a public key from a file. It takes parameters n, username, and pbfile.
use gmp scan to take in n and the username.

Create a void function to create a new private key d. It takes in parameters d, pq, p, and q.
Calculate the inverse of n mod lambda(pq) = lcm(p - 1)(q - 1).

Create a void function to write a private key to a file. It takes parameters pq, d, and pvfile.
Print pq and d using hex strings.

Create a void function to read a private key from a file. It takes parameters pq, d, and pvfile.
Use gmp scan to read the hex.

Create a void function to perform encryption. It takes parameters c, m, and n.
Pass the given parameters into the pow mod function.

Create a void function to encrypt the given infile, writing to the outfile. It takes parameters infile, outfile, and n.
The value of the encryption block must be more than 1 and less than n. Calculate the block size k with the given formula.
Dynamically allocate a block to hold k bytes.
Set the zero most byte to 0xFF.
Set j to the bytes actually read.
Utilizing a while loop, read at most k-1 bytes from the infile, placing the bytes into the allocated block from index 1.
    Use an import function to convert the read bytes.
    Encrypt with the encrypt function and write to the outfile.
Clear and close files.

Create a void function to perform decryption. It takes parameters m, c, d, and pq.
Pass the given parameters into the pow mod function.

Create a void function to decrypt the given infile, writing to the outfile. It takes parameters infile, outfile, pq, and d.
Calculate the block size k with the given formula.
Dynamically allocate a block to hold k bytes.
Utilizing a while loop, read the lines of the infile.
    Scan in the hex, saving to a variable c.
    Decrypt c to variable m, then with an export function, turn m into bytes. These are stored to the block.
    Write j - 1 bytes to the outfile.
Clear and close files.

# 4 keygen.c

**Purpose:** This file contains the main function for generating keys. It utilizes command-line arguments in order to set specifications. It generates the encryption and decryption keys. It opens and writes in the key files.
**Pseudocode:**

Define OPTIONS to be "b:i:n:d:s:vh".

The following will all be within the main().
    Initialize int opt.
    Create variables for the command line options with arguments. The defaults are used if they are not specified in the command-line.

    Create a while loop with getopt.
    Case b sets the number of bits desired for n.
    Case i sets the number of Miller-Rabin iterations.
    Case n sets the public key file.
    Case d sets the private key file.
    Case s sets the seed for random.
    Case v enables the verbose output.
    Case h displays a help message with usage.

    Open the key files if they aren't already and set the correct file permissions.
    Initialize the random state with the seed variable.
    Create the keys with the corresponding functions.
    Get the user's name.
    Write the created keys to their files.
    If verbose was enabled, print the username, p, q, n, d, and pq.
    Close the files and clear variables.

Return 0.

# 5 encrypt.c

**Purpose:** This file contains the encryption main function. It utilizes command-line arguments in order to set specifications. It encrypts the given message using the key.
**Pseudocode:**

Define OPTIONS to be “i:o:n:vh”.

The following will all be within the main().
Initialize int opt.
Create variables for the command line options with arguments. The defaults are used if they are not specified in the command-line.

Create a while loop with getopt.
Case i sets the input file.
Case o sets the output file.
Case n sets the public key file.
Case v enables the verbose output.
Case h displays a help message with usage.

Open the public key file and read the public key.
If verbose if enabled, print the username and public key n.
Encrypt the file with the corresponding function.
Close the files and clear variables.
Return 0.

# 6 decrypt.c

**Purpose:** This file contains the decryption main function. It utilizes command-line arguments in order to set specifications. It decrypts the given message using the key.
**Pseudocode:**

Define OPTIONS to be “i:o:n:vh”.

The following will all be within the main().
Initialize int opt.

Create variables for the command line options with arguments. The defaults are used if they are not specified in the command-line.

Create a while loop with getopt.
Case i sets the input file.
Case o sets the output file.
Case n sets the private key file.
Case v enables the verbose output.
Case h displays a help message with usage.

Open the key files and read the private key.
If verbose if enabled, print pq and the private key d.
Decrypt the file with the corresponding function.
Close the files and clear variables.
Return 0.

# 7 Makefile

**Purpose:** This is a file that is used to clean, format, and compile the entire program.