Assignment 5 DESIGN Document

Sam Leger

February 26, 2023

Professor Long
CSE13S Winter Quarter

1 Description of Program

This program contains three program files: a key generator, an encryptor, and a decryptor. There are three modules in the repository that have relevant functions. This program is used to conduct public-key cryptography. Public-key cryptography uses pairs of related keys, which are computed by one-way mathematical functions. The math behind cryptography is NP, meaning it is difficult to compute but is easy to verify when given an answer. The strength of this cryptography lies in the difficulty of factoring extremely large numbers.

2 Key Generator

The key generator makes the public and private keys using `ss_make_pub()` and `ss_make_priv()`. First, command-line options are handled, where the user can specify options like the number of bits for the public key, the iterations for testing primes, or the seed used for the random generator. The user can also specify the files for the private and public keys to be written to. Once options are parsed, the username is pulled from the environment and then the `ss_write_pub()` and `ss_write_priv()` functions are called. This writes the keys to their respective files.

3 Encryptor

The encryptor starts by parsing command line options. The user can specify the input file (to be encrypted), the output file (to send the encrypted message), and the name of the public key file (for reading). After options are handled, the memory for the username is dynamically allocated using `limit.h's LOGIN_NAME_MAX`. The public key file is then opened and read, using the `ss_read_pub()` function. The file is encrypted using `ss_encrypt_file()`. Finally, the input and output files are closed, and all `mpz_t` variables are cleared.

4 Decryptor

Like the previous program, the decryptor starts by looping through command line options. The user can specify the input file (to be decrypted), the output file (to send the decrypted message), and the name of the private key file (for reading). The program reads from the private key file using `ss_read_priv()` and is then able to decrypt the file, using `ss_decrypt_file()`. After the file is decrypted, input and output files are closed and `mpz_t` variables are cleared.

5 Randstate

`Randstate.c` is a small module that creates a delete a random state variable, called `state`. This gmp object is used to create the pseudorandom numbers used to make primes in `numtheory.c`.

- `gmp_randstate_t state;`
- declaration of the global variable, "state"
- `randstate_init()`
- initializes the global random state using a Mersenne Twister algorithm.
- `randstate_clear()`
- frees all memory used by the random state.

6 Numtheory

`Numtheory.c` contains the functions that drive the mathematics behind cryptography. Since the Schmidt-Samoa algorithm requires using integers that are over 64 bits (defaulting to 256), we need to employ the use of the `GMP Library`. This library allows us to create and manipulate extremely large integers.

- `pow_mod()`
- performs modular exponentiation.
- `is_prime()`
- conducts the Miller-Rabin primality test. This primality test will never report false negatives (saying a prime is a composite), but will occasionally report a false positive (calling a composite a prime). Since this has about a 25% chance of happening, we run the primality test repeatedly. This function takes two arguments, `n` and `iters`. `iters` represents the number of times we will run the test.
- `make_prime()`

- generates a new prime number, then tests it using the `is_prime()` function. If the new number is not prime, it is scrapped and a new one is generated.
- `gcd()`
- computes the greatest common divisor of provided arguments.
- `mod_inverse()`
- computes the inverse of a mod n, setting to 0 if none is found.

7 SS Library

`ss.c` contains the functions to be used directly in `keygen.c`, `encrypt.c`, and `decrypt.c`. These functions include the creation of private and public keys, their respective read and write functions and the functions for encryption and decryption.

- `ss_make_pub()`
- creates p, q, and n (p * p * q) - parts of the new public key.
- `ss_write_pub()`
- writes a public SS key, consisting of n and the username, to the public key file.
- `ss_read_pub()`
- reads the public SS key from the public key file.
- `ss_make_priv()`
- creates a new SS private key, d, given primes, and the public key.
- `ss_write_priv()`
- writes the private SS key, consisting of pq and d with trailing newlines, to the private key file.
- `ss_read_priv()`
- reads the private SS key from the private key file.
- `ss_encrypt()`
- computes ciphertext c by encrypting the message m with the public key n. This encryption is simply calling `pow_mod(c, m, n, n)`.
- `ss_encrypt_file()`

- encrypts the file in blocks. This means dynamically allocating each block, reading it in, and using `mpz_import()` to convert each block to an mpz integer. The integer is encrypted using `ss_encrypt()`, then printed to the outfile.

- `ss_decrypt()`

- computes message m by decrypting the ciphertext c with private key d and public modulus n. This decryption is simply calling `pow_mod(m, c, d, pq)`.

- `ss_decrypt_file()`

- decrypts the file in blocks. Each block is dynamically allocated. The infile is then scanned using gmp_fscanf() and decrypted using ss_decrypt(). After that, it is written to the outfile.

8 Credit and Resources

The primary resources used were lecture slides 12- *Dynamic Memory,* and 15- *Cryptography*. The C Programming Language Book, chapters 4- *Functions and Program Structure* and 6- *Structures* contained invaluable information in this assignment. The official GMP manual: `https://gmplib.org/manual/`, was necessary for the completion of `numtheory.c`.