Assignment 5 DESIGN.pdf

Julian Chop

March 1, 2023

1 Brief Description of Assignment

In this assignment, we are going to be creating three programs:

1. **keygen**: This program will generate SS public and private key pairs.
2. **encrypt**: This program will use the generated public key to encrypt a given file.
3. **decrypt**: This program will take the files encrypted files and decrypt them using the corresponding private key.

We will also have to implement 2 libraries and a random state module which will all be used in the programs mentioned earlier. I will explain more about these libraries and module further in their own sections.

The security of SS relies on large integers. And because C does not natively support arbitrary precision integers, we will also be using the GNU multiple precision arithmetic libraries, GMP for short.

A lot of this design is based on the pseudo-code/instructions from resources.

2 randstate.c

This file will contain a small random-state module utilizing GMP. This will hold a single external declaration to a global random state variable called state. We do this so other files can use this variable. This file also has 2 functions:

void randstate_init(unint64_t seed)

This function initializes the global random state named state, using the parameter seed as the random seed.

```
call srandom() using seed as param;
initialize random state using gmp_randinit_mt();
set an inital seed value into state with gmp_randseed_ui using seed as param;
```

void randstate_clear(void)

This function clears and frees all memory used by the initialized global random state called state. To do this, I can simply call the gmp_randclear(state) function.

3 numtheory.c

This file will contain the functions that drive the mathematics behind SS. These functions will be based on the given pseudo-code given in resources.

void pow_mod(mpz_t out, mpz_t base, mpz_t exponent, mpz_t modulus)

This function computes base raised to the exponent power modulo modulus. This value is then stored in out.

```
b = 1
p = base
while exponent > 0:
    if exponent is odd:
        v = (v * p) mod modulus
    p = (p * p) mod modulus
    base = base / 2
\\going to be using mpz functions
```

bool is_prime(mpz_t n, mpz_t iters)

This function conducts the Miller-Rabin primality test to check if n is prime using iters number of iterations.

```
write n-1 = 2r such that r is odd
for i in range (1, iters):
    a = random number from (2, n-2)
    y = power-mod(a,r,n)
    if y != 1 and y!= n -1:
        j = 1
        while j <= s - 1 and y != n - 1:
            y = power-mod(y,2,n)
            if y == 1:
                return false
        if y != n - 1
            return false
return true
```

void make_prime(mpz_t p, mpz_t bits, mpz_t iters)

This function generates a new prime number stored in p. This new prime number should be at least bits number of bits and primality of it should be tested iters number of times using is_prime().

```
while p isn’t a prime number:
    get random number [2^(bits-1), 2^(bits) - 1]
```

void gcd(mpz_t d, mpz_t a, mpz_t b)

This function computes the greatest common divisor of a and b and puts that value into d.

```
while b != 0:
    t = b
    b = a modulus b
    a = t
d = a
```

void mod_inverse(mpz_t i, mpz_t base, mpz_t a, mpz_t n)

This function computes the inverse i of a modulo n. If the modular inverse cannot be found, set i to 0.

```
\\the ’ represents the inverse of that variable
r, r’ = n, a
t, t’ = 0, 1
while r’ != 0:
    q = r / r’
    r , r’ = r’, (r - q * r’)
if r > 1:
    set i to 0
set s to t
\\will use auxiliary variables for the parallel assignments
```

4 ss.c

This file contains the implentation of the SS library. These functions will be used in the keygen, encrypt, and decrypt programs.

void ss_make_pub(mpz_t p, mpz_t q, mpz_t n, uint64_t nbits, uint64_t iters)

This function creats parts of a SS public key: 2 large prime numbers p and q, and n computed as p*p*q.

```
do{
    p_bits = random num [nbits/t,(2*nbits/t)]
    q_bits = nbits - (2 * p_bits)

    p = random prime number with p_bits
    q = random prime number with q_bits

}while(q-1 is divisible by p and p-1 is divisible by q)
```

void ss_write_pub(mpz_t n, char username[ ], FILE *pbfile)

This function writes a public SS key to pbfile. n is the format of the key and username is the username.

```
print n to file with newline
print username to file with newline
```

void ss_read_pub(mpz_t n, char username[ ], FILE *pbfile)

This function will read the given file that contains the public key and username

```
scan first line of file and put it in n
scan second line of file and put it in username
```

void ss_make_priv(mpz_t d, mpz_t pq, mpz_t p, mpz_t q)

This function makes a private SS key given primes p and q and the public key n.

```
d = modular inverse of (lcm(p-1, q-1)
```

void ss_write_priv(mpz_t pq, mpz_t d, FILE *pvfile)

This function writes a private SS key to pvfile. n is the format of the key and username is the username.

```
write pq to pvfile with trailing newline
write d to pvfile with trailing newline
```

void SS_read_priv(mpz_t pq, mpz_t d, FILE *pvfile)

This function reads a public SS key from pbfile. n is the format of the key and username is the username.

```
read first line of file and put it in pq
read second line op file and put it in d
```

void ss_encrypt(mpz_t c, mpz_t m, mpz_t n)

This function performs the SS encryption. Just got to implement E(m) = c = mn̂ (mod n)

```
c = m^n modulus n
```

void ss_encrypt_file(FILE *infile, FILE *outfile, mpz_t n)

This function will encrypt the contents of infile. Because we are working with modulo n, we have to encrypt the file in blocks where the value of the block of data is less than n. Furthermore, the block value cannot be 0 or 1.

```
k = block size;
allocate an array that holds k bytes with malloc();
set 0th byte of block to 0xFF to workaround byte we need;
while file is not fully read:
    read at most k - 1 bytes from file;
    place read bytes into allocated array (block +1) not to overwrite 0xFF;
    use mpz_import() to convert bytes to mpz_t m;
    encrypt block with ss_encrypt();
    write encrypted block to outfile as hexstringx;
```

void ss_decrypt(mpz_t m, mpz_t c, mpz_t d, mpz_t pq)

This function will perform the SS decryption, decrypting ciphertext c using private key d and public key d. I can do this by implementing D(c) = m = cd (mod pq).

```
m = c*d (mod pq)
```

void ss_decrypt_file(FILE *infile, FILE *outfile, mpz_t pq, mpz_t d)

This function will decrypt the contents of infile and write the decrypted contents to outfile. We gotta decrypt the same blocks as it was encrypted.

```
allocate an array that holds k bytes using malloc();
for line in file:
    scan in hexstring as mpz_t c; //each hexstring is a line
    decrypt c back into original value;
    use mpz_texport() to convert c to bytes;
    store c into the allocated block;
    write out elements read into outfile;
```

5 keygen.c

This file is going to contain the main() function for the keygen program as well as its command-line options. It will use functions from ranstate.c and ss.c to create an SS public and private key pair.

```
parse command-line options using getopt();
if -h was called:
    print help
    return;
open files for the public and private keys with fopen();
set write/read permissions of private key file to user only w/ fchmod()/fileno();
use randstate_init() to initialize random state;
use ss_make_pub() and ss_make_priv() to make key pairs;
get user's name as a string using getenv();
write keys to respective files using ss_write_pub() and ss_write_priv();
if verbose command-line option enabled:
    print out info;
```

6 encrypt.c

This file is going to contain the main() function fro the encrypt program as well as its command-line options. I will use function from ss.c to take a given file and encrypt it with the given public key.

```
parse command-line options using getopt();
if -h was called:
    print help
    return;
open public key file using fopen();
read public key from file with ss_read_pub();
if verbose command-line option is enabled:
    print out public key and username;
encrypt given file with ss_encrypt_file();
close public key file and clean and variables used;
```

7 decrypt.c

This file is going to contain the main() function for the decrypt program as well as its command-line options. It will use functions from ss.c to take an encrypted file and decrypt it with it's given private key pair.

```
parse command-line options using getopt();
if -h was called:
    print help
    return;
open private key file using fopen();
read private key from file with ss_read_priv();
if verbose command-line option is enabled:
    print out priv key and pq;
decrypt given file with ss_decrypt_file();
close private key file and clean any variables used;
```