Nayeel Imtiaz
11/11/21
CSE 13S

Assignment 6 Design Document

Summary:

This lab is all about Cryptography, which is everywhere in our life. We will have to make a key generator, an encryptor, and a decryptor, so required files include randstate.c, numtheory.c, rsa.c, keygen.c, encrypt.c, and decrypt.c. Any person can encrypt a message using the intended receiver's public key, but a private key is required to decrypt the message back. Private keys are usually held only by the intended receiver, so only they can decrypt the message. Public keys and private keys involve the RSA algorithm, and the mathematics of RSA depend on the modulo operation. The algorithm involves factoring two extremely large prime numbers (p and q). Also, a power mod algorithm is required for the key generator file. An important note is that the gmp library is required to complete this assignment. Many of the numbertheory.c functions required using only gmp code. The encryptor file should be able to encrypt input files into output files, and decryptor files should be able to decrypt input files into output files. Encrypting a file and then decrypting it back should get you the original file again.

Pseudocode:

Numtheory.c

```
gcd(d, a, b):
    Perform gcd of 'a' and 'b' and store the result into d.

mod_inverse(i, a, n):
    Perform mod_inverse of 'a' and 'n' and store the result into i.

pow_mod(out, base, exponent, modulus):
    Perform the power modulus algorithm with the numbers stored
    in base, exponent, and modulus. Then store the result in 'out'.

is_prime(n, iterations):
    Perform the Miller-Rabin algorithm.
    Return false if the number is definitely not prime.
    Return true if the number is most likely prime.

make_prime(p, x, iterations):
    Make a prime number that is at least x bits and store it into p.
```

Rsa.c

```
void_rsa_make_pub(p, q, n, e, nbits, iters):
    Make 2 big prime numbers p and q.
    Multiply p and q to get n.
    Keep generating random numbers until one
    of them is coprime with n - 1.

void_rsa_write_pub():
    Write the public RSA key to outfile using the gmp print function.

void_rsa_read_pub():
    Read the public RSA key from a file using the gmp scan function.

void_rsa_make_priv(p, q, d, e):
    Compute the mod inverse of e and (p-1)(q-1).
    Then store the result to d.

void_rsa_write_pub():
    Write the private RSA key to outfile using the gmp print function.

void_rsa_read_pub():
    Read the private RSA key from a file using the gmp scan function.

rsa_encrypt():
    Encrypt a message using the power modulo function.

rsa_encrypt_file():
    While file is not at the end:
        rsa_encrypt()

rsa_decrypt():
    Decrypt a message using the power modulo function.

rsa_decrypt_file():
    While file is not at the end:
        rsa_decrypt()
```

Randstate.c

```
randstate_init(x):
    Creates a random state with a seed of x.

randstate_clear(r):
    Deletes random state “r”.
```

Keygen.c

```
Parse command-line options using getopt()
If h flag is true:
    Show program synopsis and exit program.

Open the public and private key files using fopen()

Initialize the random state using randstate_init(seed)

Make the public and private keys using rsa_make_pub() and rsa_make_priv()
Get username using getenv().
Convert username to mpz_t.

Open private file and public file to write public and private key information.
If verbose is on, print stats.
Close all files and clear all mpz_t variables created.
```

Encrypt.c

```
Parse command-line options using getopt()
If h flag is true:
    Show program synopsis and exit program.

Read the public key from the file using rsa_read_pub().
If verbose is on, print public key information.

Open the input and output files using fopen()
Input file should be in read mode while output file is in write mode.

Convert username to mpz_t.
Verify signature using rsa_verify().
Encrypt file using encrypt_file().

Close all files and clear all mpz_t variables created.
```

Decrypt.c

```
Parse command-line options using getopt()
If h flag is true:
    Show program synopsis and exit program.
```

Read the private key from the file using rsa_read_priv().
If verbose is on, print private key information.

Open the input and output files using fopen()
Input file should be in read mode while output file is in write mode.

Decrypt file using decrypt_file().

Close all files and clear all mpz_t variables created.