DESIGN - Assignment 6

Brian Nguyen

November 10, 2021

1 Description

This assignment contains 3 programs that implement the RSA public-key cryptosystem with the structure of a key generator, encryptor, and decryptor. The key generator produces an RSA private and public key pair. The encryptor uses the public key to encrypt files, and the decryptor uses the private key to decrypt the encrypted file. These programs together are used to securely encrypt and decrypt messages or files sent between clients.

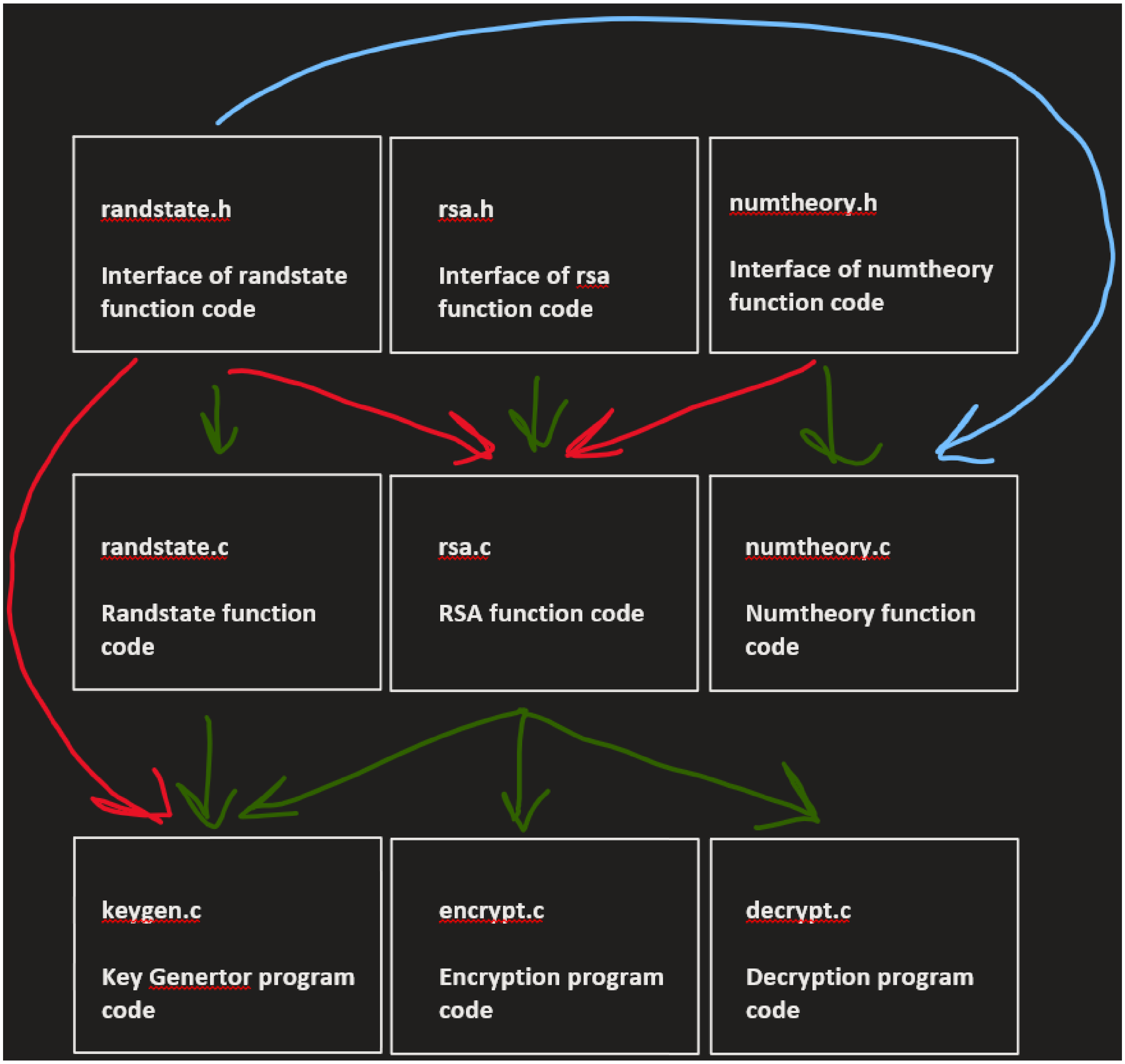
2 Files

1. decrypt.c
   - This source file contains the code for the decryption program and contains a main().
2. encrypt.c
   - This source file contains the code for the encryption program and contains a main().
3. keygen.c
   - This source file contains the code for the key generator program and contains a main().
4. numtheory.c
   - This source file contains the code for the number theory functions.
5. randstate.c
   - This source file contains the code for the random state module used in numtheory and rsa.
6. rsa.c
   - This source file contains the code for the RSA library.
7. numtheory.h

- This header file contains the interface for the number theory functions.

8. randstate.h
   - This header file contains the interface for initializing and clearing the random state.
9. rsa.h
   - This header file contains the interface for the RSA library.
10. Makefile
    - This make file contains the code that builds and compiles the program(s) to be run. It also cleans all compiler generated files and formats the code to be submitted.
11. README.md
    - This markdown file describes the program, how to build it, how to run it, and also lists and explains all the command-line options that the program accepts. It also documents any false positives given by scan-build.
12. DESIGN.pdf
    - This pdf is the manual that explains the program, files included, layout or structure, and pseudo-code of the program.

3 Structure



4 Pseudocode

4.1 Key Generator

```
define main
    initialize public key file
    initialize private key file
    use getopt() #OPTIONS: b,i,n,d,s,v,h
```

```
parse through input #use switch cases
#use fopen()
open public and private key files (rsa.pub and rsa.priv by default)
if fopen() does not work
    print error of failure and exit
if help is input
    print help message
use fchmod() with fileno() #to set permissions for priv key file
set state and seed(from input) #use randstate_init()
make public key #use rsa_make_pub()
make private key #use rsa_make_priv()
initialize char array username #will hold username
set username to getenv() #to get username as string
set username to mpz (specify base of 62) #use mpz_set_str()
write public key to public key file #use rsa_write_pub()
write private key to private key file #use rsa_write_priv()
if verbose is true
    print out username, s, p, q, n, e, d and their bits if applicable
    #s = signature, p = 1st largest prime, q = 2nd largest prime, n = pub modulus
    #e = public exponent, d = private key
close public and private key files #use fclose()
clear/free using randstate_clear() #to clear the state memory
clear/free mpz_t vars
```

4.2 Encryptor

```
define main
    intialize infile
    initialize outfile
    initialize public key file
    use getopt #OPTIONS: i,o,n,v,h
    parse through input #use switch cases
    #use fopen()
    open infile, outfile, and public key file (default: stdin, stdout, rsa.pub respectively)
    if fopen() does not work
        print error of failure and exit
    if help is input
        print help message
    read public key #use rsa_read_pub() function
```

```
if verbose is input
    print username, s, n, e and their bits if applicable
    #s = signature, n = pub modulus, e = pub exponent
initialize char array username #will hold username
convert username to mpz_t #use mpz_set_str()
verify username #use rsa_verify()
if rsa_verify() is false
    print error and exit program
encrypt to outfile from infile #use rsa_encrypt_file()
close infile, outfile, public key file #use fclose()
clear/free mpz_t vars
```

4.3 Decryptor

```
define main
    initialize infile
    initialize outfile
    initialize private key file
    use getopt #OPTIONS: i,o,n,v,h
    parse through input #use switch cases
    open priv key file (rsa.priv by default) #use fopen()
    if fopen() does not work
        print error of failure and exit
    if help is input
        print help message
    read private key #use rsa_read_priv()
    if verbose is input
        print n, e and their bits
        #n = pub modulus, e = priv key
    decrypt to outfile from infile #use rsa_decrypt_file()
    close infile, outfile, private key #use fclose()
    clear/free mpz_t vars
```

4.4 Number Theory

NOTE: We use the gmp library with mpz_t which are essentially 1 element arrays or pointers. Because of this, the use of temp or auxiliary variables (which will not be displayed in the pseudocode) must be used to prevent unexpected manipulation of input parameters (which are not specified as being changed) as we may be modifying its value from the address directly. Also, that various built-in C logic must be replaced with mpz function calls such as mpz_cmp(x,y) > 0 rather than x > y

```
#Implement the function for modular exponents to efficiently find the power of a number
#Computes base^exponent mod modulus, then puts value in out
define pow_mod #input: out, base, exponent, modulus
    set var out to 1
    while loop exponent > 0
        if exponent is odd
            set out to (out*base) mod modulus
        set base to (base*base) mod modulus
        set exponent to exponent/2
    return v

#Implement prime checker
#Tests if a number p is prime via the Miller-Rabin Algorithm based on iters iterations
define is_prime #input: n, iters
    check for 1, return true
    check for 3, return true
    check for n mod 2 = 0
        if 2, return true
        else, return false
    set r to n-1
    while loop r is even
        divide r by 2
        increment s by 1
    for loop i=1 to iters
        choose random a #from range 2 to n-2
        set y to power_mod of y,a,r,n
        if y is not 1 and n-1
            set j to 1
            while loop j <= s-1 and y is not n-1
                set y to power_mod of y,2,n
                if y is 1
                    return false
                increment j by 1
            if y is not n-1
                return false
    return true

#Implement prime maker
```

```
#Generates a random number p that is bits long, then tests if its prime using is_prime
define make_prime #input: p, bits, iters
    while loop is_prime of p not true and not bits long
        choose random p number
    return p

#Implement GCD Euclidean algorithm
#Finds the greatest common divisor between a and b
define gcd #input: d, a, b
    while b is not 0
        set temp to b
        set b to a mod b
        set a to temp
    set d to a
    return d

#Implement Modular Inverse of number
#Finds the modular multiplicative inverse of a (mod n) and stores in i
define mod_inverse #input: i, a, n
    set r to n
    set r_prime to a
    set i to 0
    set i_prime to 1
    while r_prime is not 0
        set q to r/r_prime
        set temp to r
        set r to r_prime
        set r_prime to temp-(q*r_prime)
        set temp to i
        set i to i_prime
        set i_prime to temp-(q*i_prime)
    if r > 1
        set i to 0
        return i
    if i < 0
        set i to i + n
    return i
```

4.5 Random State

```
#Generates random arbitrary-precision integers for programs that use it
define randstate_init #input: seed
    initialize global var state #from extern var in header
    call gmp_randinit_mt() #with state var
    call gmp_randseed_ui() #with seed var

#Clear and Free memory from the state
define randstate_clear
    call gmp_randclear() #use state var
```

4.6 RSA Library

NOTE: We use the gmp library with mpz_t which are essentially 1 element arrays or pointers. Because of this, the use of temp or auxiliary variables (which will not be displayed in the pseudocode) must be used to prevent unexpected manipulation of input parameters (which are not specified as being changed) as we may be modifying its value from the address directly. Also, that various built-in C logic must be replaced with mpz function calls such as mpz_cmp(x,y) > 0 rather than x > y

```
#Implement RSA function that generates public key
#Generates a public key for the user made of random prime (p), random prime (q),
#product of p*q (n), and random public exponent (e)
define rsa_make_pub #include: p, q, n, e, nbits. iters
    call make_prime() for p #use iters
    call make_prime() for q #use iters
    choose random nbits_prime in range [nbits/4, (3*nbits)/4]
    set nbits_prime to p
    set nbits-nbits_prime to q
    set n to (p-1)(q-1)
    set e to random number #nbits ish long, using mpz_urandomb()
    while gcd(e,n) != 1
        randomize e again

#Implement RSA function to write public key to pbfile
#Writes the generated public key to the pbfile in specific formatted output
define rsa_write_pub #input: n, e, s, username[], *pbfile
    print n with new line to pbfile
    print e with new line to pbfile
    print s with new line to pbfile
```

```
    print username with new line to pbfile
    #use gmp_fprintf() for mpz and fprintf for username

#Implement RSA function to read public key from pbfile
#Reads the generated public key from the pbfile in specific formatted input
define rsa_read_pub #input: n, e, s, username[], *pbfile
    scan n with new line from pbfile
    scan e with new line from pbfile
    scan s with new line from pbfile
    scan username with new line from pbfile
    #use gmp_fscanf() for mpz and fscanf for username

#Implement RSA function to generate private key
#Generates the private key based on p, q, e from rsa_make_pub and stores in d
define rsa_make_priv #input: d, e, p, q, *pvfile
    #e, p, q are given
    set d = ((p-1)(q-1)) mod e

#Implement RSA function to write private key
#Writes the generated private key to pvfile in specific formatted output
define rsa_write_priv #input: n, d, pvfile
    print n with new line to pvfile
    print d with new line to pvfile
    #use gmp_fprintf()

#Implement RSA function to read private key
#Reads the generated private key from the pvfile in specific formatted input
define rsa_read_priv #input: n, d, pvfile
    scan n with new line from pvfile
    scan d with new line from pvfile
    #use gmp_fscanf()


#Implement RSA function to encrypt message
#Encrypts message and stores in var c using pow_mod from numtheory
define rsa_encrypt #input: c, m, e, n
    set c to m^e (mod n) #using numtheory functions
```

```
#Implement RSA function to encrypt a file
#Encrypts the infile in k sized blocks at a time and writes to outfile in hexstrings
define rsa_encrypt_file #input: infile, outfile, n, e
    set k to floor division of ((log base 2 of n) - 1)/8
    dynamically allocate array block of size k #using uint8_t size chunks, calloc()
    set block at index 0 to 0xFF

    initialize m to hold encrypted message
    initialize j to be bytes read from infile
    while loop (j = read block sized chunk from infile) > 0 #use fread()
        call mpz_import() #put in mpt_z m, with order = 1, 1 endian = 1, and nails = 0
        #and other in-function vars like j
        encrypt m from import #usse rsa_encrypt()
        print encrypted m with new line to outfile

#Implement RSA function to decrypt message
#Decrypts message and stores in var m using pow_mod from numtheory
define rsa_decrypt #input: m, c, d, n
    set m to c^d (mod n) #using numtheory functions

#Implement RSA function to decrypt file
#Decrypts the infile by scaning hexstrings and placing it in k sized blocks
#used to write the decrypted file to outfile
define rsa_decrypt_file #input: m, s, e, n
    set k to floor division of ((log base 2 of n) - 1)/8
    dynamically allocate array block of size k #using uint8_t size chunks, calloc()

    initialize c to hold decrypted message
    initialize j to be bytes read from infile
    while loop (scan hexstring from infile put in c) > 0 #use gmp_fscanf()
        decrypt c using n,d #use rsa_decrypt()
        call mpz_export() to put in block #take from c, use same parameters as encrypt file
        #and other in-function vars like j
        write from block to outfile #use fwrite()

#Implement RSA function to sign message
#Signs the message and stores in var s using pow_mod in numtheory
define rsa_sign #input: s, m, d, n
```

```
    set s to m^d (mod n) #using numtheory functions

#Implement RSA function to verify message
#Verifies the signed message using an expected message m using pow_mod in numtheory
define rsa_verify #input: m, s, e, n
    set t to s^e (mod n) #use numtheory functions
    if m = t
        return true
    return false
```

5 Error Handling

1. The prime seeker can only probably find a prime with uncertainty, but definitely find if a number is not prime.
2. The product n of p and q != nbits all the time (sometimes it is short by 1 bit) so 1 is added to both bits(p) and bits(q) to make sure it is greater than or equal to nbits as needed by the asgn6 pdf.
3. Some edge cases may not have been accounted for, such as getting username (or if its NULL), opening files, hitting RAND_MAX if bits are too large, etc.

6 Credits

1. I used the asgn6.pdf from Professor Long for explanations and pseudocode.
2. I watched Eugene's lab section recordings held on 11/9.