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 key decryption program and contains a main().
2. encrypt.c
   - This source file contains the code for the key 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 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 pseudocode of the program.

3 Structure

4 Pseudocode

4.1 Key Generator

```
define main
    use getopt #OPTIONS: b,i,n,d,s,v,h
    use switch cases #to parse through input
    use fopen() #to open pub and priv key files (rsa.pub and rsa.priv by default)
    if fopen() does not work
        print error of failure and exit
    use fchmod() #to set and make sure permissions are correct
    use fileno() #to set and make sure permissions are correct
    use randstate_init() #to set state and seed (from input)
    use rsa_make_pub() #to make pub key
    use rsa_make_priv() #to make priv key
    use getenv() #to get username as string
```

```
    use mpz_set_str() #to set and make sure permissions are correct (specify base of 62)
    use rsa_write_pub() #to write pub key
    use rsa_write_priv() #to write priv key
    if verbose is true
        print out username, s, p, q, n, e, d
        #s = signature, p = 1st largest prime, q = 2nd largest prime, n = pub modulus
        #e = pub exponent, d = priv key
    use fclose() #to close pub and priv key files
    clear/free using randstate_clear() #to clear the randstate
    clear/free mpz_t vars
```

4.2 Encryptor

```
define main
    intialize infile
    initialize outfile
    use getopt #OPTIONS: i,o,n,v,h
    use switch cases #to parse through input
    use fopen() #to open pub key file(rsa.pub by default)
    if fopen() does not work
        print error of failure and exit
    use gmp fscanf() #to read pub key
    if verbose is input
        print username, s, n, e
        #s = signature, n = pub modulus, e = pub exponent
    use gmp fscanf() #to read username and convert it to mpz_t
    use rsa_verify() #to verify username
    if rsa_verify() is false
        print error and exit program
    use rsa_encrypt_file() #to encrypt infile to outfile
    use fclose() #to close pub key file
    clear/free mpz_t vars
```

4.3 Decryptor

```
define main
    initialize infile
    initialize outfile
    use getopt #OPTIONS: i,o,n,v,h
    use switch cases #to parse through input
```

```
    use fopen() #to open priv key file (rsa.priv by default)
    if fopen() does not work
        print error of failure and exit
    use gmp fscanf() #to read priv key
    if verbose is input
        print n, e
        #n = pub modulus, e = priv key
    use rsa_decrypt_file() #to decrypt infile to outfile
    use fclose() #to close priv key
    clear/free mpz_t vars
```

4.4 Number Theory

```
#Implement the function for modular exponents to efficiently find the power of a number
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
define is_prime #input: n, iters
    write n-1=(2^s)r # r is odd, use loop to generate s and r, given n
    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
define make_prime #input: p, bits, iters
    while loop is_prime of p not true
        choose random p number #n-bits long
    return p

#Implement GCD Euclidean algorithm
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
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

```
#Set the 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
define randstate_clear
    call gmp_randclear() #use state var
```

4.6 RSA Library

```
#Implement RSA function that generates public key
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
define rsa_write_pub #input: n, e, s, username[], *pbfile
    call gmp_fprintf(pbfile, var) #use hexadecimal rational format specifier
    #do for n, e, s, username

#Implement RSA function to read public key from pbfile
define rsa_read_pub #input: n, e, s, username[], *pbfile
    call gmp_fscanf() #use same specifier
    store in respective variables

#Implement RSA function to generate private key
define rsa_make_priv #input: d, e, p, q, *pvfile
    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 d = ((p-1)(q-1)) mod e

#Implement RSA function to write private key
define rsa_write_priv #input: n, d, pvfile
    call gmp_fprintf(pbfile, var) #use hexadecimal rational format specifier
    #do for n, d

#Implement RSA function to read private key
define rsa_read_priv #input: n, d, pvfile
    call gmp_fscanf() #use same specifier
    store in respective variables

#Implement RSA function to encrypt message
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
define rsa_encrypt_file #input: infile, outfile, n, e
    set k to block size #floor division of ((log base 2 of n) - 1)/8
    dynamically allocate array of k bytes #using uint8_t size chunks
    set byte at index 0 to 0xFF
    call gmp_fscanf() #from infile, starting at index 1 to k (at most)
    set j to bytes read from infile
    put bytes actually read in k #using loop from 1 - j-bytes read
    call mpz_import() #put in mpt_z m, with order = 1, 1 endian = 1, and nails = 0
    #(and other in house variables)
    call rsa_encrypt() to encrypt m to outfile

#Implement RSA function to decrypt message
define rsa_decrypt #input: m, c, d, n
    set m to c^d mod n #using numtheory functions

#Implement RSA function to decrypt file
define rsa_decrypt_file #input: m, s, e, n
    set k to block size #floor division of ((log base 2 of n) - 1)/8
```

```
    dynamically allocate array of k bytes #using uint8_t size chunks
    call gmp_fscanf() #save bytes from infile to mpz_t c
    call mpz_export() #put mpt_z c into j, with order = 1, 1 endian = 1, and nails = 0
    #(and other in house variables)
    write to outfile from 1 - j-1 bytes

#Implement RSA function to sign message
define rsa_sign #input: s, m, d, n
    set s to m^d mod n #using numtheory functions

#Implement RSA function to verify message
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.

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.