DESIGN.pdf ASGN6

Nathan Ong

November 11, 2021

1 Description

This collection of files contains an encryption program used to encrypt public files using a public key, a decryption program used to decrypt those public files using a corresponding private key, and a key generator that generates public and private keys. All three programs utilize an RSA library, several number theory functions, and random state variables, all of which will be covered in Section 3 (Pseudocode and Structure).

The encrypt file used to encrypt public files has the following command line options:

- -i *infile* Specifies input file. Default input file is *stdin*.
- -o *outfile* Specifies output file. Default output file is *stdout*.
- -n Specifies public key file. Default file is *rsa.pub*.
- -v Enables verbose output.
- -h Prints out help message then exits the program.

The decrypt file used to encrypt public files has the following command line options:

- -i *infile* Specifies input file. Default input file is *stdin*.
- -o *outfile* Specifies output file. Default output file is *stdout*.
- -n Specifies private key file. Default file is *rsa.priv*.
- -v Enables verbose output.
- -h Prints out help message then exits the program.

The key generator used to generate the public and private keys has the following command line options:

- -b Specifies minimum bits needed for public modulus.
- -i Specifies number of Miller-Rabin iterations for testing primes. Default number is 50.
- -n *pbfile* Specifies public key file. Default file is *rsa.pub*.
- -d *pvfile* Specifies private key file. Default file is *rsa.priv*.
- -s Specifies random seed for random state variables. Default seed is the seconds since the UNIX epoch (time(NULL)).
- -v Enables verbose output.
- -h Prints out help mesage then exits the program.

2 Files Included in the Directory

1. decrypt.c
   This file contains the implementation of the decryption program.

2. encrypt.c
   This file contains the implementation of the encryption program.

3. keygen.c
   This file contains the implementation of the key generator program.

4. numtheory.c
   This file contains the implementation of the number theory functions.

5. numtheory.h
   This file contains the specification of the interface for the number theory functions.

6. randstate.c
   This file contains the implementation of the random state interface for the number theory functions and RSA library.

7. randstate.h
   This file contains the specification of the interface for initializing and clearing the random state.

8. rsa.c
   This file contains the implementation of the RSA library.

9. rsa.h
   This file contains the specification of the interface for the RSA library.

3 Pseudocode and Structure

3.1 decrypt.c

while opt isnt -1
    indice through arguments
    check for required arguments if necessary
open private key file
    print error message if failed
read private key from file
if verbose output is enabled print verbose information
decrypt file
close file and clear mpz t variables

3.2 encrypt.c

while opt isnt -1
    indice through arguments
    check for required arguments if necessary
open public key file

print error message if failed
read public key from file
if verbose output is enabled print verbose information
convert username into mpz t
verify signature
encrypt file
close file and clear mpz t variables

3.3 keygen.c

while opt isnt -1
    indice through arguments
    check for required arguments if necessary
open public and private key files
    print error message if failed
set private key file permissions to 600
initialize random state and seed
make public and private keys
get current users name
convert username to mpz t with base 62
compute signature of username
write public and private keys to respective files
if verbose output is enabled print verbose information
close files and clear random state and mpz t variables

3.4 numtheory.c

gcd:
d = divisor, a = first number, b = second number
while b is not 0
    store b as temporary variable
    b = a mod b   a = temporary variable (old b)
return a

mod inverse:
n = modulus number, a = number to get inverse of, t = temporary variable, t' = 2nd temporary variable
r = n, r' = a, t = 0, t' = 1
while r' is not 0
    q = floor(r / r')
    r = r'
    r' = r - (q x r')
    t = t'
    t' = t - (q x t')
if r greater than 1, return no inverse
if t less than 0, t = t + n
return t

pow mod:
n = modulus number, p = base number, d = exponent number
while d is greater than 0
    if d is odd, v = (v x p) mod n
    p = (p x p) mod n
    d = floor(d/2)

return v

is prime:
k = number of iterations, n = number to test for primality
write n - 1 = 2^sr such that r is odd
for 1 to k
    choose random number a between 2 and n - 2
    y = pow mod(a, r, n)
    if y is not 1 and y is not n - 1
        j = 1
        while j is less than or equal to s - 1 and y is not n - 1
            y = pow mod(y, 2, n)
            if y = 1 return false
            increment j by 1
        if y is not n - 1 return false
return true (number is prime)

make prime:
nbits = generated random number length in bits, k = number of iterations of is prime test, p = stored random number
generate random number that is nbits bits long, store in p
for 1 to k
test primality with is prime()
if number is prime, return p

3.5 randstate.c

randstate init:
initialize global variable state
call gmp randint mt()
call gmp randseed ui()

randstate clear:
call gmp randclear to free and clear memory

3.6 rsa.c

rsa make pub:
create 2 primes using make prime() compute e mod phi(n) = (p - 1)(q - 1)
loop generate numbers of around nbits using mpz urandomb()
compute gcd of each random number and the totient
stop when random number is a coprime of the totient

rsa write pub:
write public key to pbfile
(key format should be n, e, s, username, in that order, with a trailing newline, with n, e, and s being hexstrings)

rsa read pub:
read public key from pbfile
(key format should be n, e, s, username, in that order, with a trailing newline, with n, e, and s being hexstrings)

rsa make priv:
generate new private key with (e mod phi(n) = (p - 1)(q - 1))

rsa write priv:
write private key to pvfile
(key format should be n followed by d, which should both be hexstrings)

rsa read priv:
read private key from pvfile
(key format should be n followed by d, which should both be hexstrings)

rsa encrypt:
encrypt ciphertext using (E(m) = c = m^e (mod n)

rsa encrypt file:
calculate block size with floor(log_2(n) - 1)/8)
allocate array that can hold a block size's worth of bytes
set zeroth byte to 0xFF
while there are unprocessed bytes
    read a maximum of k - 1 bytes from infile
    place read bytes into allocated array starting from index 1   convert read bytes into an mpz t using mpz import, with order parameter 1, endian parameter 1, and nails parameter 0
    encrypt new mpz t with rsa encrypt
    write encrypted number into outfile as a hexstring, with a trailing newline

rsa decrypt:
decrypt ciphertext using (D(c) = m = c^d (mod n)

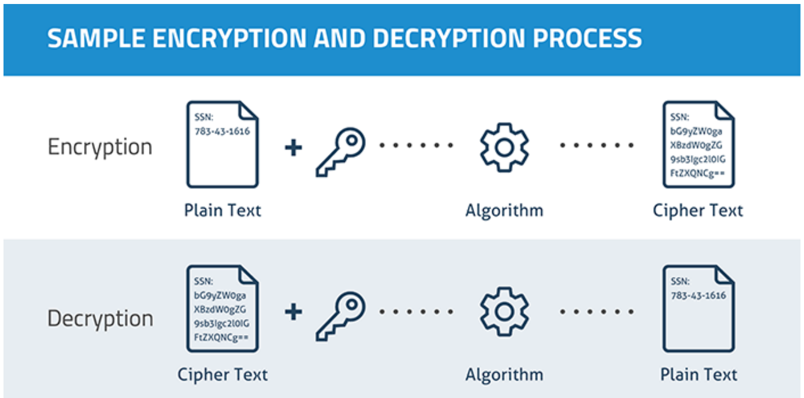
rsa decrypt file:
calculate block size with floor(log_2(n) - 1)/8)
allocate array that can hold a block size's worth of bytes
while there are unprocessed bytes
    scan hexstring, save as mpz t
    convert mpz t back into bytes, and store in array using mpz export, with order parameter 1, endian parameter 1, and nails parameter 0
    write maximum of j - 1 bytes into outfile
    place written bytes into allocated array starting from index 1

rsa sign:
return key signature (S(m) = s = m^d (mod n))

rsa verify:
return true if signature (t = V(s) = s^e (mod n)) is verified
return false if otherwise

4 Additional Diagram

4.1 Simplified Graph of Encryption and Decryption



5 Additional Credits

1. The Makefile format was given by Sloan in his in-person section on October 5th.

2. Images use Creative Commons License CC BY-SA 4.0.

   (a) CC BY-SA 4.0
   https://creativecommons.org/licenses/by-sa/4.0/deed.en

3. Image Credits:

   - Encryption and Decryption Graph:
     Title: Sample Encryption and Decryption Process
     Author: Munkhzaya Ganbold
     Date Created: 1 Nov 2017