Dmitrius Agoston
Prof. Darrell Long
20 November 2021

CSE 13S Fall 2021
Assignment 6: Public Key Cryptography
Design Document

Description of Program:
The purpose of this program is to create a key that will be used for the encryption and decryption of files. This process is done through three main functions. Those being, the creation of a key using a key generator, and then both the encryption and the decryption of the desired files using the generated key. First the key generator is given the task of creating RSA public and private key pairs. With these key pairs generated, the encrypter uses the public key in order to encrypt the given files. The decrypter can then use the corresponding private key to decrypt the previously encrypted files.

Layout/Pseudocode:

- randstate.c
  This file is responsible for the creation and use of random, arbitrary precision integers. The functions in this file will be used with the GMP library since C does not natively support the use of arbitrary precision integers that are needed in order for RSA to function. Overall, this file will initialize a random state variable that can be used with the necessary GMP functions. Along with this, there is also a function that clears and frees any memory used in the initialization of the random state.

  Define randstate_init with seed
      Call gmp_randinit_mt and use seed
      Call gmp_randseed_ui and use seed

  Define randstate_clear
      Call gmp_randclear

- numtheory.c
  This file is responsible for all of the number theory behind the operations that will be used in the RSA library being created. In particular, modular exponentiation, primality testing, and modular inverses will be necessary for the RSA library. First off, modular exponentiation is necessary since RSA requires the computation of “a” to the power of “n”. If the numbers being used were much smaller this would be no problem, except, as mentioned earlier, RSA requires arbitrary precision

integers. Meaning that modular exponentiation is necessary in order for the program to run much more efficiently. With primality testing, the test that will be used in this case is the Miller-Rabin primality test. While this test is not perfect, it is good enough for the uses in this program. The test is able to determine whether or not a number is prime, except the test is only accurate 75% of the time. Normally this would be a problem, but if the test is run 100 times then the chance of being wrong decreases significantly. For modular inverses the function that is used will give the modular inverse of a number “a” modded by another number “n”. Though this function can also determine if there is no modular inverse.

```
Define pow_mod with out, base, exponent, modulus
    Initialize p, is_odd, e, and v
    Set v to 1
    Set p to base
    Set e to exponent
    Loop while e is greater than 0
        Mod e by 2 and give value to is_odd
        Check if the result above is odd
            Multiply v by p
            Mod v by modulus
        Multiply p by itself
        Mod p by modulus
        Divide e by 2
    Set out to v
    Clear initialized local mpz

Define is_prime with n, iters
    Check for special cases of 0 through 3
    Initialize s, r, t, m, a, y, dec, decr, and two mpz
    Set t equal to n - 1
    Set two to 2
    Loop until odd
        If odd
            Set r to t
        If even
            Increment s
            Divided t by 2
    Loop through number of iterations desired
        Set a to n - 3
```

Set a to a random number
Increment a by 2
Power mod y, a, r, and n
If y does not equal 1 and y does not equal dec
Set t to 1
Set decr to s - 1
Loop while t is less than or equal to decr and y does not equal dec
Power mod y, y, two, and n
If y equals 1
Clear mpz and return false
Increment t
If y does not equal dec
Clear mpz and return false
Clear mpz and true

Define make_prime with p, bits, iters
Generate a new random prime number with a min of bits
Set prime number to p
Call is_prime with p and iters to check if prime or not

Define gcd with d, a, b
Initialize t, ta, and tb mpz
Set ta to a
Set tb to b
Loop while tb is less than 0
Set t to tb
Set tb to ta mod tb
Set ta to t
Set d to ta
Clear initialized local mpz

Define mod_inverse with i, a ,n
Initialize r, rr, t, and tt mpz
Set r to n
Set rr to a
Set tt to 1
Loop while rr does not equal 0
Initialize q, temp, and tempo mpz
Set q to r divided by rr

Set temp to rr
Set tempo to tt
Set r to q - rr
Set t to q - tt
Set rr to r
Set tt to t
Set r to temp
Set t to tempo
Clear initialized local mpz
Set i to t
Check if r is greater than 1
Set i to zero since there is no inverse
Check if t is less than 0
Set i to t + n
Clear initialized local mpz

- rsa.c
  This file is the library that is responsible for the creation of the RSA public and private key pairs. There are functions in this library that are made for handling the public and private keys. These include making, reading, and writing functions for both public and private keys. There are also functions for general encryption and decryption along with actual file encryption and decryption. Functions for the RSA signature and verification have also been included in this file.

  Define rsa_make_pub with p, q, n, e, nbits, iters
  Initialize range, coprime, totient, and rand mpz
  gSet range to 3 * nbits / 4 - nbits / 4
  Set p to random number with range
  Add nbits / 4 to p
  Set q to nbits - p
  Make p and q a prime
  Set n to p * q
  Set totient to p - 1 * q - 1
  Loop until coprime equals 1
  Set rand to a random number
  Get the gcd of rand and totient and store in coprime
  Set e to rand
  Clear local mpz

  Define rsa_write pub with n, e, s, username, pbfile

Write n, e , and s as a hexstring to the pbfile
Write username to the pbfile

Define rsa_read_pub with n, e, s, username, pbfile
Read n, e, and s as a hex string
Read username of public key

Define rsa_make_priv with d, e, p, q
Compute the inverse of e modulo (p-1)(q-1)
Set d equal to the computer inverse

Define rsa_write_priv with n, d, pvfile
Write n as hexstring to pv with newline
Write d as hexstring to pv with newline

Define rsa_read_priv with n, d, pvfile
Read n and d as a hex string from pvfile

Define rsa_encrypt with c, m, e, n
Pow mod c, m, e, and n

Define rsa_encrypt_file with infile, outfile, n, e
Initialize c and m mpz
Create variable k and set to log base 2 of n - 1 / 8
Create block and allocate memory
Set position 0 of block to 0xFF
Create variable j and set to 1
Loop until j equals 0
Set j equal to fread of block + 1
Call mpz import
Encrypt c, m, e, n
Write c to the outfile
Clear local mpz
Free memory allocated to block

Define rsa_decrypt with m, c, d, n
Pow mod m, c, d, and n

Define rsa_decrypt_file with infile, outfile, n, d
Initialize mpz c

Create variable k and set to log base 2 of n - 1 / 8
Create block and allocate memory
Create j and set equal to 0
While j does not equal 0
Set c to 0
Read infile
Decrypt c, c, d, and n
Export block
Print everything in block
Clear local mpz
Free memory allocated to block

Define rsa_sign with s, m, d, n
Pow mod s, m, d, and n

Define rsa_verify with m, s, e, n
Initialize t
Pow mod t, s, e, and n
Check if t is equal to m
Return true and clear t
Return false and clear t

- keygen.c

  This file contains all of the functions that are necessary for the generation of key pairs to be used in the encryption/decryption process. The key generator is capable of receiving command line inputs in order to help specify the creation of the key. There is an option to input the minimum bits required when doing the public modulus of “n”. Another option is to designate how many iterations should occur in the Miller-Rabin test for primes. There is also the option to designate the file to be used for both the public and private key that will be generated. If the user desires there is an option for a seed that will be used in the random number generation in order to create reproducible results. Along with this there is an option to enable verbose printing for more data and if the user needs help with the program there is a default help message for the basic usage and synopsis.

  Define main
  Parse through command line
  If h entered
  Print help statement
  End program

If v entered
Enable verbose printing
If b entered
Set minimum number of bits to value given
If n entered
If key file is valid
Set public key to given key file
If d entered
If key file is valid
Set private key to given key file
If s entered
Set seed to given value for seed
Check the permissions of the private key file
Initialize random state through randstate_init()
Make both public and private keys
Use getenv() to get the user's name
Convert the username into an mpz_t and then set signature
Write the public and private keys to corresponding files
If verbose printing is enabled
Print username, signature, first large prime, second large prime, public modulus, public exponent, and private key
Close files and free memory

- encrypt.c
This file contains the functions that are used in the process of encrypting a given file. There are options that the user can change in order to get an encryption that they desire. The first of these options is to choose which exact file to encrypt and where to store the output of the newly encrypted file. There is also the choice for which public key to use and if no key is given a default one will be used. For more data there is a verbose option to print to the output along with an option for a help message that describes the general usage and synopsis of the program.

Define main
Parse through command line
If h entered
Print help statement
End program
If v entered
Enable verbose printing
If n entered

                If public key file is valid
                    Set public key file to given public key file
            If i entered
                If file to encrypt is valid
                    Set infile to given file to encrypt
            If o entered
                If file to write encryption to is valid
                    Set outfile to given file to write to
        Read public key from the from public key file
        If verbose printing enabled
            Print username, signature, public modulus, and public exponent
        Convert username into mpz_t and use rsa_verify to verify signature
        Encrypt the given file
        Close files and free memory

- decrypt.c

  This file contains the functions that are used in the decryption process of an already decrypted file. Much like encrypt there are similar options given for the decryption process. The first is the option to specify the input file to the decrypter and where to put the output of the decrypted file. There is also the option to give a private key for the decryption process and if not specified a default key will be used. The option for verbose printing is also available along with the option to print a help statement to help the user with the usage and understanding of the program.

```
Define main
    Parse through command line
        If h entered
            Print help statement
            End program
        If v entered
            Enable verbose printing
        If n entered
            If private key file is valid
                Set private key file to given private key file
        If i entered
            If file to decrypt is valid
                Set infile to given file to decrypt
```

If o entered
If file to write decryption to is valid
Set outfile to given file to write to
Read private key from the from private key file
If verbose printing enabled
Print modulus and private key
Decrypt the file
Close files and free memory

Error Handling:
When handling errors for this program, the main errors occurred in regards to user/file input. There had to be checks for valid files being used for both encryption and decryption along with valid private and public key files. Along with that there had to be permission given when a key needs to be generated. Some other errors that had to be handled involved how memory was being allocated. The program has to appropriately allocate memory for certain variables and at the end of running has to free up the space it used for those variables. This includes having to free the space used for the random state as well as all of the mpz variables that were used.

Credit:
When creating this code I largely used asgn6.pdf as my main source of reference for creating this program.