Assignment 5 DESIGN.pdf
Yash Sharma

Description of Program:

This assignment requires us to implement three programs: a key generator, encryptor, and decryptor. The key generator program will be in charge of key generation, producing RSA public and private key pairs. The encryption program will encrypt files using a public key, and the decryption program will decrypt the encrypted files using the corresponding private key. In the assignment, we will also need to implement two libraries and a random state module that will be used for each of these programs. Out of these two, one library should focus on holding functions relating to the mathematics behind RSA, while the other should contain implementations of routines for RSA. In addition, we will need to utilize a library that can be portrayed in other words as the GNU multiple precision arithmetic libraries.

Files to be included in directory “asgn5”:

1. decrypt.c:
    - This file contains the implementations of the decrypt program.
2. encrypt.c:
    - This file contains the implementation of the encrypt program.
3. keygen.c:
    - This file contains the implementation of the keygen program.
4. numtheory.c:
    - This file contains the implementation of the number theory functions.
5. numtheory.h:
    - This file contains the interface for the number theory functions.
6. randstate.c:
    - This file contains the implementation of the random state interface for the RSA library and number theory functions.
7. randstate.h:
    - This file contains 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 interface for the RSA library.
10. Makefile:

- A file that formats program into clang-format and compiles it into program executables with make/make all from Makefile
- Additionally, make clean from Makefile must remove compiler-generated files (such as the executables)

11. README.md:
    - This file describes how to use the program and MakeFile. It will also list and explain any command-line options the program accepts.
12. DESIGN.pdf:
    - This describes the design for the program thoroughly with pseudocode and descriptions.

Pseudocode / Structure:

decrypt.c

Include header files

Create a usage function that contains all the options for the user to select when running the program

Create a main function that takes in argc and argv as parameters

Initialize variables for input, output, prive, and opt

Create a while loop that parses through getopt with switch cases for each option within the decrypt.c file

Initialize variables n, d using the MPZ Library

Open the private key file

Read private key

If the input is not equal to the output

    Open input file for reading

    If infile contains nothing

        Return an error message

        Close file

        Clear MPZ variables

        Return an exit failure statement

Call a function to read the private key file

If the verbose output is true,

    Print public modulus n

    Print private key e

If the input is not equal to the output

    Open output file for writing

    If infile contains nothing

        Return an error message

        Close file

        Clear MPZ variables

        Return an exit failure statement

Decrypt the file using rsa decrypt file

Close private key files and clear MPZ variables

encrypt.c

Include header files

Create a usage function that contains all the options for the user to select when running the program

Create a main function that takes in argc and argv as parameters

Initialize variables for input, output, public key, and for the verbose case

Create a while loop that parses through getopt with switch cases for each option within the encrypt.c file

Initialize the character username value to base 256

Open public key file with fopen and read the file

Set infile to the standard file

If the input is not equal to the file

    Open the input file and read the file

    If the output is not valid

Print an error message

Terminate loop

Initialize the mpz variables

Read the public key from the public file

If verbose output is true,

Print username, signature s, the public modulus n, and public exponent e

Convert username read to mpz_t and verify signature using rsa verify

Set a file equal to standard output

If the output is not valid

Open the output file for writing

If the output file is not valid

Print an error message

Clear the MPZ variables

End the loop

Encrypt file using rsa encrypt file

Close public key

Clear MPZ variables

keygen.c

Include header files

Create a usage function that contains all the options for the user to select when running the program

Create a main function that takes in argc and argv as parameters

Initialize variables for minimum bits, number of miller-rabin iterations, specifying the public key, specifying the private key, specifying the random seed, enabling verbose output, and displaying usage

For input and output in the cases, set those equal to optarg

Create a while loop that parses through getopt with switch cases for each option within the keygen.c file

Get the username of the user in the form of a character

Create a getenv function to get the current user's name in a string

Open public and private keys with fopen

Initialize MPZ variables

Set private key permissions to 0600 using fchmod and fileno

Initialize randstad init using the set seed

Create the rsa make pub and rsa make priv functions

Convert the username using mpz set str and rsa sign to compute the signature

Write the computed public and private key files

If verbose output is true,

> Print username, the signature s, the first large prime p, the second large prime q, the public modulus n, the public exponent e and the private key d.

Close files

Clear random state and MPZ Variables

Numtheory.c

Include header files

Create a function that calculates the greatest common divisior of a & b and stores the value in a new variable called b.

Create a while loop that checks if b is not equal to 0

> If so then set t equal to b
>
> Set b equal to a modulus b
>
> Set a equal to t

Return the greatest common divisor of a and b

Create a function for the modular inverse of two parameters: a and b

> Set n and a equal to r and r prime
> Set 0 and 1 eqiual to t and t prime

While r prime is not 0,

> Set q equal to r divided by r prime
> Set r and r prime equal to r prime and r minus q time r prime
> Set t and t prime equal to t prime and t minus q times t prime

If r is less than 1,

> Return that there is no inverse

If t is less than 0,

    Set t equal to t plus n

Return inverse modulus value t

Create a power modulus function that takes in out, base, exponent, and modulus

Initialize values for the 3 variables

Create a while loop that checks if one variable is greater than 0

    If odd, set the one variable equal to temp

    Multiply the other two variables and set them equal to the same variable

    Set the temp variable equal to the first variable

Return the final value by set the tmpa equal to d

Clear MPZ variables

Create an is_prime function that checks the miller rabin test for a prime variable, this function should take two parameters: n and iters

Write a variable that checks if r is odd in n-1 = 2^s*r

Check if n is equal to 0 and if that value is an even integer

    If so, then clear the MPZ variables

Return false

Check if n = 1,2,3 is equal to 0

    If so, then clear the MPZ variables

Return true

Set a while loop that loops through conditions if r is an even number

    While looping, add 1 to s

    Divide r by a temp variable

Set and subtract temp variables temp2 and n

Create a while loop that checks when iters is greater than 1

    When looping, set a equal to the random state of the variable

    Add 2 to a

Call the power mod function

If y = 1 and y = temp2

Set j equal to 1

While j is less than the temp variable,

Call the pow mod function

If y = 1

Clear the MPZ variables

Return false

Add the output back to j

Check if y equals temp2

If so, then clear mpz variables

Return false

Clear MPZ variables

Return true

Create a make_prime function to make a variable prime in a list p

Initialize the MPZ variables

Set a variable t equal to 2

Set the exponent equal to t to the power of bits

While the p and iters call the isprime function, then run through a loop

Set p to a random var

Add the prime number back to the random variable

Clear MPZ variables

return

Randstate.c

Include header files

Initialize and create a randstate function that takes in a seed integer for a parameter

Call the random state function for this seed

Make calls to the GMP functions

Clear and free all memory stored by the random state

Call the GMP function to clear the data

rsa.c

Include header files

Create an LCM function that can be used to calculate the least common multiple for a prime

Create the rsa make public key using parameters p, q, n, e, t, nbits, and iters

- Initialize MPZ variables
- Calculate pbits
- Calculate qbits
- Make qbits and pbits into prime numbers using iters
- Calculate the product of two prime numbers
- Calculate the size of a block using MPZ
- Make a while loop that runs to make numbers prime when x is less than nbits
  - Call make prime function using qbits and iters
  - Multiply p and q together and output to n
  - Find the sizeinbase of the n
- Make q into a prime number
- Subtract p and q from 1
- Call the lcm function to compute the least common multiple between p1 and q1
- Multiply the product of the two prime numbers p and q
- Compute the greatest common divisor of e and 1
- While gl is equal to 1
  - Compute the greatest common divisor
- Clear MPZ variables

Create the write public RSA key function using n, e, s, username, and pbfile

- Print n, e, s and username

Create a function that reads the public RSA key using the same parameters

- Scan n, e, s, and username

Create a function that creates a new RSA private key using parameters d, e, p, and q

- Initialize MPZ variables
- Subtract p and 1, output to p1
- Subtract q and 1, output to q1
- Call the LCM function to find the least common multiple between p1 and q1
- Calculate the inverse modular of e and 1
- Output value in hexstring
- Clear MPZ variables

Create a function that writes a new RSA private key using parameters n, d, and pvfile

- Print n using pvfile
- Print d using pvfile

Create a function that reads a new RSA private key using parameters n, d, and pvfile

- Scan n using pvfile
- Scan d using pvfile

Create a function that encrypts an rsa key using parameters c, m, e, and n

- Call the pow mod function for c, m, e, and n

Create a function that encrypts the contents of infile, writing the encrypted contents to outfile using parameters infile, outfile, n, and e

- Initialize MPZ variables
- Calculate the block size of k
- Create heap memory array using malloc
- Read from the heap and calculate the bytes read
- Create a while loop that checks if the numBytesRead are greater than 0
    - If so, then import bytes read and output value to m
    - Call rsa encrypt and encrypt the RSA Message
    - Print message from the outfile
    - Read from the heap and calculate the bytes read
- Free heap
- Clear MPZ variables

Create a function that decrypts an rsa key using parameters m, c, d, and n

Call the pow mod function for m, c, d, n

Create a function that decrypts the contents of infile, writing the encrypted contents to outfile using parameters infile, outfile, n, and d

Initialize MPZ variables

Calculate block size of k

Create heap memory and store it on the stack

Create a while loop that scans infile and writes decrypted messages back to the heap

Call rsa decrypt to decrypt the message

Export block message from the heap onto block

Write the block message onto outfile

Free heap

Clear MPZ Variables

Create a function that performs RSA signing, producing a signature by signing messages. This function uses s, m, d, and n as parameters

Call power modulus function for s, m, d, n

Create a function that verifys RSA, returning true itf the signature s is verified and false otherwise. This function utilizes m, s, e, and n as parameters. This function also uses a signature that is verified if and only if t is the same as the expected message m.

Initialize the MPZ variables

Call power modulus for t, s, e, n and calculate the modulus of an exponent

If t is equal to m

Clear MPZ variables

Return true

Else

Clear MPZ variables

Return false

Credit:

- I attended Eugene's section on 2/4/21, which helped give me general guidance on how to approach this lab, as well as sketch out how the program runs.

- I used the pseudocode from the asgn5 documentation.
- I also used the asgn5 documentation and reviewed the pseudocode provided by Elmer in discord.