Alex Asch
Assignment 6 Design
Description of Program
Deliverables

-
- decrypt.c
    - This file contains the implementation RSA decryption.
- encrypt.c
    - This file contains the implementation RSA encryption.
- keygen.c
    - This file contains the implementation of RSA key generation.
- numtheory.c
    - This file contains the implementations of the number theory functions.
- numtheory.h
    - This file specifies the interface for the number theory functions.
    - This file is provided in the resources repository by Professor Long. (2021 Professor Long)
- randstate.c
    - This contains the implementation of the random state interface for the RSA library
- randstate.h
    - This specifies the interface for initializing and clearing the random state.
    - This file is provided in the resources repository by Professor Long. (2021 Professor Long)
- rsa.c
    - This contains the implementation of the RSA library.
- rsa.h
    - This specifies the interface for the RSA library.
    - This file is provided in the resources repository by Professor Long. (2021 Professor Long)
- Makefile
    - Makes the above program.
    - Links the above files.
- README.md
    - A document in markdown syntax on the usage of the program.
- DESIGN.pdf
    - The pdf being viewed at the moment (this document).

Notes

- **Random State**

- Using a seed initializes a random number generator using the Mersenne Twister Algorithm.
    - Set the seed as something passed into the function.
    - Have a function to clear the random state as well.
    - Uses gmp library functions
- **Modular exponentiation**
    - Uses mpz_t types for arguments and result.
    - Computes a base raised to exponent power modulo modulus, and stores the result in the output
    - Also have a function to take the modular inverse.
- **Testing primality**
    - This is done using a probabilistic algorithm, the Miller Rabin method.
    - Use the initialized random number generator here.
    - To make a random prime, iterate through random numbers until one that is found is prime.
- **GCD**
    - Use the Euclidean method for calculating the greatest common denominator.
- **RSA lib**
    - First create the parts of a RSA key, these are the two large primes, the public exponent, and the private key.
    - Use the above number theory methods to do this.
    - **RSA encrypt**
        - Read in a file.
        - Read in a public key.
        - Go through the file and encrypt each block of text using the public key, import the text as an mpz_T to do arithmetic operations on the block of text.
        - Write the computed numbers to an output file.
    - **RSA decrypt**
        - Read in numbers from an input file.
        - Read in a private key.
        - Decrypt each number from the input file using the private key.
        - Convert the number into strings of bits. Output these to the output file.

Pseudocode

- randstate_init
    - Initialize the external state variable.
    - Seed the external state with the given variable.
- randstate_clear
    - Clear the external state variable.
- pow_mod

- Does fast modular exponentiation, base raised to the exponent power module modulus and storing the computed result in out.
- Make temporary variables as needed to.
- While the variable d is greater than 0.
  - If d is odd.
    - Set v to itself times p modulo n.
  - Set p to p squared modulo n
  - Divide d by two.
- Return the value in v.
- Because gmp passes a pointer, set a equal to v for returning the value.

- is_prime
  - First, find and r and s such that n(being the number to test) - 1 = 2^s*r and r is odd.
    - To do this, make a while loop such that until an odd r is found, divide r (starting at n - 1) by 2, starting at division by one.
    - Have a counter variable starting at zero count each iteration.
    - Once an odd r is found s is the counter variable and r is r.
  - For i in range of 1 to k(inclusive)
    - Generate a random number, modulo (n-4) +2, this makes a random number between 2 and n -2.
    - Set y equal to y squared modulo n.
    - If y is not 1 or n - 1
      - Start a variable j at one.
      - While j is less than s minus one, and y is not equal to n -1.
        - Set y equal to y squared modulo n.
        - If y equals 1.
          - This number is not prime, return false.
      - If y does not equal n -1 return false, this number is not prime.
  - If all above loops finish iteration without hitting a point to return false, y is prime, return true.
- make_prime
  - Generates a random prime number of around n bits long.
  - Create a prime, check if it is prime.
  - If not prime, generate and check again, iteratively.
  - If it is, return that number./
- gcd
  - Takes the greatest common denominator of a and b, store it in d and return it.
  - While b is not zero.
    - A is equal to b, and b is equal to a mod.
  - Return the value in a through d.

- mod_inverse
  - Take i and a and n, compute the modular inverse of a mod n.
  - Pseudo will be written assuming parallel assignment is allowed, every time we do so, in reality a temporary variable is used.
  - Set r to n, and r’ to a.
  - Set t to, and 0 t’ to 1
  - While r’ is not 0.
    - Set q to r/r’.
    - Set r to r, and r’ to r - q*r’.
    - Set t to t’, and t’ to t-q*t’.
  - If r is greater than 1.
    - Set i to zero for no inverse found.
  - If t is less than 0.
    - t is equal to t +n.
  - Return t through i.
- rsa_make_pub
  - Creates two new primes, p and q, the bits for p will be a random number in the range, number of bits/4 and 3 number of bits/4. The bits for q will be the number of bits minus the bits for p.
  - Next compute the totient of pq, which is (p-1)(q-1).
  - While we haven’t found the prime exponent yet.
    - Generate a new random number.
    - Is the gcd of totient and the random number 1?
      - Then we found the prime exponent, stop iteration.
- rsa_write_pub
  - Write the public key values, then the username to the outfile.
  - Write each with a newline after, and write the numbers as hexstrings.
- rsa_make_priv
  - Given two primes and the public exponent, compute the private key. The private key is the public exponent inverse modulo the totient of p and q.
- rsa_write_priv
  - Write n then d, both with trailing newline, and as hexstrings.
- rsa_read_priv
  - Reads a private key from a file, the format of a private key should be n then d, read both of these values, and store them as mpz_t
- rsa_encrypt
  - Takes a message as a mpz_t m, then compute the ciphertext as c = m^e (mod n).
- rsa_encrypt file
  - Encrypts the contents of input file to the output file using a given public file with public modulo n.

    - Step one, set a block size var k = floor of log2(n-1)/8.
        - Dynamically allocate an array that can hold k bytes.
        - Set byte one of the array to 0xFF.
        - Read from the infile until the rest of the array is full, or end of file is reached.
        - Using mpz_import, convert the buffer to an mpz_t.
        - Encrypt this number using the above encrypt.
        - Take the encrypted number, write it, then a newline.
        - Repeat until the end of file.
- RSA decrypt
    - Compute a message from ciphertext, and a private key.
    - The message is the cipher text to the power of d modulo n.
- rsa_decryptor
    - Calculate block size, as the floor of log2(n-1)/8.
    - Make an array that can hold k bytes to hold the block.
    - Read a line from the encrypted file as an mpz_t.
        - Decrypt the line with the above function.
        - Turn the decrypted number into an array of bytes.
        - Print all but the first byte in that array.
        - Repeat until there are no more bytes in the input file.
- rsa_sign
    - The signature is the given message to the power of d modulo n.
    - Compute this and return it.
- rsa_verify
    - If the given signature to the power of e modulo n is the same as the expected message, return true, if it is not, return false.

KEYGEN

- Parse command line arguments for the number of bits, the iterations for Miller Rabin, the seed, the key files, verbose output, and help message.
- Set the private files to 0600 permissions.
- Initialize a randstad with the seed.
- Make the public and private keys using the above functions, with the number of bits and Miller Rabin iteration count specified.
- Get the usernames and change it to mpz_t, and compute that as the message to sign with, write this with the public key.
- Write the public and private key.
- If verbose printing is enabled, print the verbose output.

ENCRYPTOR

- Parse command line inputs of input and output file, public key, verbose output, and help.
- Open the key and input file for reading, open the output file for writing.

- Read the public key from the given file.
- If verbose output is enabled, print the contents of the public key.
- Convert the read username to an mpz_t and verify it ensuring it is the same as the given message.
- Encrypt the file.

DECRYPTOR

- Take arguments for input file, output file, and private key, as well as verbose output and help.
- Open the private key and read it.
- If verbose output is specified.
    - Print the contents of the private key file.
- Decrypt the file.