Assignment 5: Design Doc

Madison Ormsby

February 17, 2022

1 Description

Using a few c files to implement number theory, the RSA library, and creating a rasndomstate interface, three executable c programs will be created. The three programs are `keygen.c`, `encrypt.c`, and `decrypt.c`. The first program will generate a public and private key duo to specified files. The second program will encrypt a message from a specified input into a specified output file. The third will decrypt a message from a specified input into a specified output.

`Numtheory.c` implements certain number theory functions using a `gmp` library. The functions are `gcd` or greatest common denominator, `mod_inverse` or modular inverse, `pow_mod` or power modulus, `is_prime` which uses the Miller-Rabin primality test to determine if the input number is prime, and `make_prime` which generates a randomly generated prime number.

2 Files

- `decrypt.c`: Contains the `main()` program for the `decrypt` program. Will accept arguments:
  - `-i`: sets the input file for `decrypt` (default is stdin).
  - `-o`: sets the output file for `decrypt` (default is stdout).
  - `-n`: sets the file that contains the private key (default is rsa.priv).
  - `-v`: enables verbose text output.
  - `-h`: displays the program's features and usage.
- `encrypt.c`: Contains the `main()` program for the `encrypt` program. Will accept arguments:
  - `-i`: sets the input file for `encrypt` (default is stdin).
  - `-o`: sets the output file for `encrypt` (default is stdout).
  - `-n`: sets the file that contains the public key (default is rsa.pub).
  - `-v`: enables verbose text output.
  - `-h`: displays the program's features and usage.
- `keygen.c`: Contains the `main()` program for the `keygen` program. Will accept arguments:
  - `-b`: specifies the minimum number of bits needed for creating prime numbers for the public modulus.
  - `-i`: specifies the number of iterations for the Miller-Rabin test.
  - `-n pbfile`: specifies the public key file (default is rsa.pub).
  - `-d pbfile`: specifies the private key file (default is rsa.priv).

  - `-s`: specifies a random seed for random state initialization (default is seconds since 00:00:00 UTC January 1, 1970).
  - `-v`: enables verbose text output.
  - `-h`: displays the program's features and usage.

- `numtheory.c`: Contains the code for the number theory functions.
- `numtheory.h`: The header file containing the interface for `numtheory.c`.
- `randstate.c`: Contains the code for the random state interface for the `rsa.h` library and `numtheory.h` functions.
- `randstate.h`: The header file containing the interface for `randstate.c`.
- `rsa.c`: Contains the code for the RSA library functions.
- `rsa.h`: The header file containing the interface for `rsa.c`.

3 Pseudocode — numtheory.c

```
gcd(output, a, b)
    initialize a temp variable
    while b doesnt equal 0
        set temp = b
        set b = a modulus b
        set a = temp
    free the temp variable
    store a in output
```

```
pow-mod(output, base, exponent, modulus)
    initialize out = 1
    initialize a = base
    while exponent ¿ 0
        if exponent is odd
            out = (out * a) mod modulus
        set a to (a * a) mod modulus
    store the out in output
```

```
is-prime(n, iters)
    initialize vars s and r such that $n - 1 = 2^s r$ and r is odd
    for i ¡ iters starting at 1
        choose a random number between 2 and n-2 and set it as a
        set a variable y equal to power-mod(a, r, n)
            if y ! = and y ! = n - 1
                set variable j = 1
                while $j <= s - 1$ and $y! = n - 1$
                    set y to power-mod(y, 2, n)
                    if y = 1
                        return false
                    add one to j
                if $y! = n - 1$
                    return false
    return true
```

```
make-prime(output, bits, iters)
    left shift one by the number of bits and set that as the minimum
    do
        create a random number
        add the minimum to the random number
        set the random number to output
    while output is not prime
```

```
mod-inverse(output, a, n)
    set r to n, and rp to a
    set t to 0, and tp to 1
    while rp ! = 0
        set q to the floor of $r/rp$
        use a temp variable to store rp and tp
        set rp to (r - q * rp)
        set tp to (t - q * tp)
        set r and t to the temp variables above
    if r -gt 1 there is no inverse
    if t -lt 0
        add n to t
    store t in output
```

4 Notes — numtheory.c

- Since we are using the `gmp` library, before starting each function, make a temporary variable for each input variable and set each input to the temp variables. After the function is done, reset the input variables to the temp variables.
- `isprime` uses the Miller-Rabin formula to calculate.

5 Pseudocode — randstate.c

create a global variable state

```
randstate-init(uint64t seed)
    initialize the random state with seed
    initialize the state for the Mersenne Twister algorithm

randstate-clear(void)
    clear out the memory from the state
```

6 Pseudocode — rsa.c

```
rsa-make-pub(p, q, n, e, nbits, iters)
    set a variable pbits to a random number mod (2* nbits / 4) + 1)) + (nbits / 4)
    set a variable qbits to the remainder of nbits - pbits
    make two primes, p and q, with makeprime size pbits and qbits
    find the greatest common denominator of (p - 1) and (q - 1)
    multiply p and q together and set n to the product
    multiply (p - 1) and (q - 1) together and set that to a variable ni
    floor divide ni by the gcd and set that to variable lcm
    do
        create a random number e nbits long
        take the greatest common denominator of e and lcm
    while the greatest common denominator of e and lcm is not = 1
```

```
rsa-write-pub(n, e, s, username, pbfile)
    write n to the public file
    write e to the public file
    write s to the public file
    write the username to the public file
```

```
rsa-read-pub(n, e, s, username, pbfile)
    scan the first line into mpz n
    scan the next line into mpz e
    scan the next line into mpz s
    scan the next line into char username
```

```
rsa-make-priv(d, e, p, q)
    make variables pt, qt, n, lcm, gc
    initialize the variables as mpz-ts
    set pt to p - 1
    set qt to q - 1
    multiply pt and qt and set as n
    find the gcd of pt and qt, and set it as gc
    floor divide n by gc and set the result to lcm
    take the mod-inverse of e and lcm and set the result to d
    clear the variables
```

```
rsa-write-priv(n, d, pvfile)
    write n to the private file
    write d to the private file
```

```
rsa-read-priv(n, d, pvfile)
    scan the first line into n
    scan the second line into d
```

```
rsa-encrypt(c, m, e, n)
    take the power mod of m, e, n and set to c
```

```
rsa-encrypt-file(infile, outfile, n, e)
    create and initialize variables m, c as mpz-ts
    create variables k and j of type size-t
    calculate $k = floor((log_2(n) - 1)/8)$
    dynamically allocate space into a block of type uint8-t
    while the end of file hasn't been reached
        set the first item of the block to 0xFF
        read at most k - 1 bytes into the block starting at index 1
        if more than 0 items are read
            convert the block into an mpz-t
            encrypt the mpz
            write the resulting message into the outfile
            set the mpz to 0 and start over
    free the block
    clear out the variables
```

```
rsa-decrypt(m, c, d, n)
    take the power mod of c, d, n and set to m
```

```
rsa-decrypt-file(infile, outfile, n, d)
    create and initialize mpz-t variables c, m, and o
    create variables k and j of type size-t
    calculate $k = floor((log_2(n) - 1)/8)$
    dynamically allocate space into a block of type uint8-t
    set the first index of the block to 0xFF
    while the end of file hasn't been reached
        scan a hexstring in and set it to c
        decrypt c using rsa-decrypt
        convert c into bytes and store into the block, set j to the number of bytes converted
        write j - 1 bytes starting from index 1
    free the block
    clear the mpz-t variables
```

```
rsa-sign(s, m, d, n)
    take the power mod of m, d, and n and set to s
```

```
rsa-verify(m, s, e, n)
    create and initialize mpz-t t
    take the pow-mod of s, e, and n and set the result to t
    if t is equal to m return true
    else return false
```

7 Pseudocode — keygen.c

```
help(void)
    print out a summary and synopsis of the function


main(argc, **argv)
    create uint64 variables bits, iters, seed, opt, and filen
    create file pointers pubfile, and privfile
    create a boolean variable verbose that is false
    create a string called user
    set seed equal to time(NULL)
    create and initialize mpz-ts p, q, n, e, s, and username
    set opt to 0
    set bits to 256
    open rsa.pub into pubfile
    open rsa.priv into privfile
    create a getopt loop to cycle through args "b:i:n:d:s:vh"
        use a switch for opt
            in case b
                bits = uint64-t optarg
            in case i
                iters = uint64-t optarg
            in case n
                close the pubfile
                set the pubfile to fopen the optarg
                if the pubfile doesn't exist return 0
            in case d
                close the privfile
                set the priv file to fopen the optarg
                if the privfile doesn't exist return 0
            in case s
                set seed to the optarg
            in case v
                set verbose equal to true
            default
                use help()
    set the file permission for privfile to 0600
    initialize the random state
    make the public key
    make the private key
    make the signature for the user
    write the public key
    write the private key
    if verbose is true
        print information on the user, s, p, q, n, e, and d
    close the public and private file
    clear the randstate
    clear the mpz variables
    return 0
```

8 Pseudocode — encrypt.c

```
help(void)
    print out a summary and synopsis of the function


  main(argc, **argv)
    create file pointers fpin, and fpout
    set fpin to stdin, and fpout to stdout
    create file pointer key and open "rsa.pub"
    create a boolean variable called verbose that is false
    create a string for the username called user
    create an integer variable opt set to 0
    create and initialize variables username, n, e, and s in mpz-t form
    create a getopt loop to cycle through args "i:o:n:vh"
        use a switch for opt
            in case i set fpin to the optarg
            in case o set fpout to the optarg
            in case n
            fclose the file pointer key and reopen it with the optarg
            if the pointer returns null return 0
            in case v set verbose to true
            default prints the help function
    read the public key from fpin into n, e, s, and user
    if verbose is true, print the user, s, n, and e
    set the mpz username to the string user in base 62
    verify the username is equal to the signature
    encrypt the file in fpin and output to fpout
    close the file pointers
    clear the mpz-ts
    return 0
```

9 Pseudocode — decrypt.c

```
help(void)
    print out a summary and synopsis of the function


  main(argc, **argv)
    create file pointers fpin and fpout
    set fpin to stdin, and spout to stdout
    create file pointer key and open "rsa.priv"
    create a boolean variable called verbose that is false
    create an integer variable opt set to 0
    create and initialize variables n and e in mpz-t form
    create a getopt loop to cycle through args "i:o:n:vh"
        use a switch for opt
            in case i set fpin to the optarg
            in case o set fpout to the optarg
    in case n
    fclose the file pointer key and reopen it with the optarg
    if the pointer returns null return 0
    in case v set verbose to true
    default prints the help function
read the private key from fpin to n and e
if verbose is true, print n and e
decrypt the file fpin and output into fpout
close the file pointers
clear the mpz-ts
return 0
```