[redacted]

Design Doc

General Idea:

1. randstate.c
    a. randstate.c : small random state module with one extern variable that represents a global random variable for the seed and two functions. One of the functions will initialize the random seed and the other one will clear the seed.
    b. The file provides the first global variable that initializes the state.
    c. Initialize the function for initializing the random state that's needed for RSA key generation operations. The function returns nothing and takes in one parameter that represents the seed.
    d. Initialize the function that frees memory that is used after initializing the random state. Called at the end of all key generation or number theory operations

Pseudocode

-Declare function for finding the totient with three parameters, one for the base 2 and another as the exponent, and the last one for n-1

-Initialize a temporary variable and set it equal to the parameter n

-Initialize a second temporary variable for the value of the first temporary variable above modulus 2

-Create a while loop that has a condition that's true when the second temporary variable equals 0

-increment the parameter value of s by one

-divide the first temporary variable by 2

-item set the second temporary variable equal to the first one modulus 2

-set the first temporary variable equal to the second parameter for the exponent

-Declare function for initializing random state with one parameter for the seed, nothing gets returned

-Call gmp library function gmp randinit mt to initialize the state for the Mersenne Twister algorithm

-Use gmp randseed ui function with the global state as a parameter and the seed passed into this function

-Declare function for clearing the random state, nothing returned and no parameters

-call gmp randclear function

2. numtheory.c
    a. numtheory.c : Library that will implement functions that drive the mathematics behind RSA
    b. Modular Exponentiation Function: function that does modular exponentiation rapidly by computing base raised to the exponent power modulo modulus, and storing this value in out (the parameter).
    c. Primality Testing Function (is prime) : Use the Miller-Rabin primality test and the specified number of iterations to indicate whether the given number is prime
    d. Primality Testing Function 2 (make prime): Creates a new prime number from the given mpz value. The number's primality should be tested using the function above this (is prime) in the number of iterations specified as a parameter
    e. Modular Inverses (greatest common divisor function): This function will compute the greatest common divisor between the two values specified as parameters and stores them in the first parameter variable
    f. Modular Inverses (mod-inverse function): Computes the inverse of a modulo given as a parameter. If the inverse cannot be calculated, it is set as 0.

Pseudocode

-Declare the power mod function with four parameters, one for the output, one for the base of the logarithm, one for the exponent, and one for the modulus value. The return type is void.

-Initialize the first parameter representing the output equal to 1

item Initialize another variable and set it equal to the parameter that represents the base

-while the value of the parameter representing the exponent is greater than 0

-if the value of the exponent is odd (calculated by seeing the remainder after dividing by 2)

-set the first variable equal to its current value multiplied by the second initialized variable modulus the the last parameter value called modulus

-set the second variable equal to its value squared modulus the last parameter value

-update the value of the exponent to its value divided by 2

-clear initialized variables

-Initialize a variable that is equal to the totient value (in helper functions section)

-Declare function for checking whether a value is prime. Two parameters, one to check primality and one for the number of iterations. Return type of boolean

-if number is 1

-return false

-if number is 2

-return true

-if number is 3
    - return true
-if number is 4
    -return true
-find the totient
-Initialize a temporary variable and set it equal to the parameter n
-Initialize a second temporary variable for the value of the first temporary variable above modulus 2
-Create a while loop that has a condition that's true when the second temporary variable equals 0
    -increment the parameter value of s by one
    -divide the first temporary variable by 2
    -item set the second temporary variable equal to the first one modulus 2
-set the first temporary variable equal to the second parameter for the exponent

-Create for loop that loops from 1 to the number of iterations specified as a parameter
    -choose a random number between two and the given parameter value minus 2
    -create a new variable y that is equal to the power-mod function with the variable above as a parameter, 2 as the second parameter, and the value of n specified in the function parameter
    -if the value y above is not equal to 1 or equal to the parameter n minus 1
        -set a variable j equal to 1
        -while the variable y is less than or equal to s - 1 and the variable with the power-mod function doesn't equal n-1
            -set the variable y equal to power-mod with parameters y, 2 and n
            -if y is equal to 1
                -return False
            -set the variable j equal to j plus 1
        - if y doesn’t equal n-1
            -return False
-return True
-clear initialized variables

- Declare a make_prime function that will make a function prime. Include three parameters, one for the resulting prime number, one for the number of bits, and one for the number of iterations
    -set a variable equal to 2 to the power of the number of bits and a variable for return value
    -set a boolean variable to false
    -while the above boolean variable is false (haven’t found prime number)
        -generate a random number that is the bits parameter long
        -add an offset of 2 to the power of bits to the random number
        -generate the next prime number
        -if the number is prime
            -set the resulting prime number equal to the random number

-set the boolean to true
-clear initialized variables

-Declare a function for gdc with three parameters that represent the resulting value and two other values that we have to get the greatest common divisor for
-initialize a variable t and two temporary variables for the value of the first and the second number for gcd
-Create while loop that continues when the value of the third parameter is not equal to 0
-set variable t equal to the third parameter value
-set the third parameter equal to the second parameter mod the third parameter
-set the second parameter equal to t
-set the return value parameter equal to the second parameter
-clear initialized variables

-Declare a function for mod_inverse with three parameters, one for the resulting value, one for the mod value and one for the actual value
-Initialize a variable r and set it equal to the third parameter value
-Initialize a second variable r' and set it equal to the second parameter value
-set a variable t equal to 0
-set another variable t' equal to 1
-while the value of the second variable is not equal to 0
-set the value of a temporary variable q equal to the floor division of r and r'
-set the value of r equal to r'
-set the value of r' equal to r minus q multiplied by r'
-set the value of t equal to t'
-set the value of t' equal to t minus q multiplied by t'
-if r is greater than 0
-return no inverse
-if t is less than 0
-set t equal to t plus the third parameter
-clear initialized variables

---

3. rsa.c
    a. rsa.c: Includes all functions needed for creating the keys involved in encryption and decryption
    b. RSA make public key: This function will have two large prime numbers, their product and the public exponent e. It will first create keys then find an exponent that will help generate random numbers that are around nbits
    c. RSA write public key: This function will write a public key to the file and format it with the username each with a trailing new line

d. RSA read public key: This function will read a public RSA key from the file and format it
e. RSA make private key: This function creates a new RSA private key given two prime numbers and a public exponent
f. RSA write private key: This function writes a private RSA key to the given file
g. RSA read private key: This function reads a private key from the file and formats it with hexstrings
h. RSA encrypt: This function performs the encryption and computes ciphertest by encrypting the message using public exponent e and modulus n
i. RSA encrypt file: This function encrypts the contents of the infile to the outfile in blocks
j. RSA decrypt: This function performs the RSA decryption with the cipherfile and private key with public modulus n
k. RSA decrypt file: decrypts contents of infile and writes the decrypted contents to outfile in blocks
l. RSA sign: this function performs the RSA signing by producing signatures and messages using private keys and the public modulus n
m. RSA verify: this function will return true if the signature is verified and false otherwise

Pseudocode

-Declare helper function for the lowest common multiple with parameters representing the two values to calculate and the resulting value

-create temporary variables
-use Carmichael's formula: |(p-1)(q-1)|/gcd(p-1, q-1)
-use mpz multiply to multiply the first and the second parameter and then get the absolute value
-get the gcd value for the same two values
-divide the numerator by the denominator with gcd
-clear all the initialized variables

-Declare the function for making the public key with 6 parameters. The first two represent the two large prime numbers (p, q) then one for their product another for the public exponent e, one for the bits (nbits), and one for the iterations (iters)

-initialize an upper variable and set it equal to 3 times nbits divided by 4
-initialize a lower variable to nbits divided by 4
-set another variable for the range which is the upper minus lower
-set another variable using urandomm with the range above
-set another variable called bits_p equal to rand()
-set another variable called bits_q equal to the nbits minus bits_p
-Create a prime number p of bits_p
-Create a prime number q of bits_q

-Set the value of n equal to p times q
-use totient function with parameters p-1 and q-1
-Create a while true loop
-create a random number that is nbits long
-if the gcd of the random number and the totient is 1
-break
-set the value of e to the random number
-clear all the initialized variables

-Declare the rsa_write_pub function with 5 parameters. Three for the public key (n,e,s) and then one for the username (username), and one for the file (*pbfile)
-initialize the variables necessary
-create temporary variables for the parameters
-Print n as a hex string with the formatter %Zx
-Print e as a hex string with formatter %Zx
-Print s as a hex string with formatter %s
-clear variables

-Declare the rsa_read_pub function with 5 parameters (n,e,s,username, *pbfile)
-Scan value of n which is a hex string and format using %Zx
-Scan value of e which is hex string with specifier %Zx
-Scan value of s which is a hex string with specifier %Zx
-Scan the username and format with %s

-Declare the rsa_make_priv function with 4 parameters, one for private key d, given primes p and q, public exponent e
-set totient to (p-1)*(q-1)
-use mod_inverse function and pass in d,e,n
-Clear variables

-Declare the rsa_write_priv function with three parameters, one for the private key (n) then d with trailing new line and the file
-create temporary variables and set temporary variables equal to the parameters
-print the value of n as a hex string with specifier %Zx
-print the value of d as a hex string with specifier %Zx
-clear variables

-Declare rsa_read_priv function with three parameters, one for the key n, then d and the file
-Scan the parameter n ad use format specifier %Zx
-Scan the parameter d and use format specifier %Zx

-Declare the rsa_encrypt function with four parameters, with ciphertext c, message (m) and public exponent r and modulus n

-use pow_mod function and pass in c,m,e,n

-Declare the rsa_encrypt_file with four parameters, infile, outfile, n and public e
-set the block size to the floor division of log base 2(n) minus 1 divided by 8
-allocate space for the uint8_t block with size plus 1
-allocate the char buffer size for the block size
-When the infile character is not NULL
-Create while true loop
-set the 0th element of the byte to 0xFF
-set the variable j equal to fread(buffer size, size of char, the block size, infile)
-place the read bytes into the allocated block starting from the second spot of the array through a while loop
-convert the read bytes to mpz_t with import()
-rsa_encrypt(c,n,r,n)
-print c as a hex string with %Zx
-if the value of j is less than the block size
-break
-clear variables
-free block and buffer

-Declare function for rsa_decrypt with four parameters (m,c,d,n)
-use pow_mod function with four parameters: m,c,d,n

-Declare function for rsa_decrypt_file with four parameters, infile, outfile, n and d
- set the block size to the floor division of log base 2 (n) minus 1 divided by 8
-allocate a uint8_t block of size k
-if the infile isn't NULL
-Create while true loop
-scan c, a hex string with %Zx
-if you're at the end of the file
-break the loop
-run rsa_decrypt with parameters m,c,d,n
-convert the c into bytes and store in block with export()
-fwrite(the array plus 1, sizeof uint8_t, j-1, outfile)
-clear variables
-free block

-Declare function for rsa_sign with four variables: s,m,d,n
-use pow_mod function with parameters s,m,d,n

-Declare rsa_verify with four parameters: m,s,e,n
-initialize variable t
- use pow_mod function with parameters t,s,e,n

- if t is equal to m
    -clear t
    -return true
-Else
    -clear t
    -return false

---

4. encrypt.c
    a. General idea: Contains implementation and main() function
        i. Implement as written on document with options: -i,-o,-n,-v,-h

Pseudocode

-Declare the main loop
    -initiate variables for verbose output and the files
    -create while loop with all the options
        -if the option is i
            -set the input file
        -if the option is o
            -set the output file
        -if the option is n
            -set the key file
        -if the option is v
            -set the verbose output to true
        -If the option is h
            -print the help option
        -set the default to the help option

    -set the temporary variables and the username
    -open the pbfile with the public file
        -print the error message
        -clear variables
    -set the infile to the in stream
        -if the file can't be opened
            -print the error message
            -clear the variables
            -return 1
    -set variables for the original message

-read the public key
-if the verbose output is set
    -print the username, the signature, modulus, and exponent

-set the message
-if the message can't be verified
    -print the error message
    -close files

-encrypt the file
-clear variables and close files

---

5. keygen.c
    a. General idea: Contains implementation and main() function
        i. Implement as written on document with options: -b,-i,-n,-d,-s,-v,-h

Pseudocode

-Declare the main loop
- initiate variables seed, iters, bits, verbose output
-initiate variables for verbose output and the files
-create while loop with all the options
    -if the option is b
        -if the value is positive set it equal to the bits
    -if the option is i
        -if the iters is positive then set that variable
    -if the option is n
        -set the public key variable
    -if the option is d
        -set the private key variable
    -If the option is s
        -set the seed
    -If the option is v
        -set the verbose output
    -If the option is h
        -print the help option
    -set the default to the help option

-open files for private and public key
-if the private file is null
    -set it executable
    -set the given file to this

-if the public file is null
    -print the error
    -return 1
-if the private file is null
    -print the error
    -return 1

-set the random state
-initiate the necessary variables
-make the public and private key
-set the username variable
-set it as a string
-set the rsa sign

-write both public and private keys
-if verbose is enabled
    -print username, signature, prime p, prime q, modulus n, exponent e, private key d

-close files, clear translate, and return 0

---

6. decrypt.c
    a. General idea: Contains implementation of number theory functions
        i. Implement as written on document with options: -i,-o,-n,-v,-h

Pseudocode

-Declare the main loop
    -initiate variables for verbose output and the files
    -create while loop with all the options
        -if the option is i
            -set the input file
        -if the option is o
            -set the output file
        -if the option is n
            -set the private key file
        -if the option is v
            -set the verbose output to true

-If the option is h
-print the help option
-set the default to the help option

-declare variables used later with d and n
-open the pvfile with the private file
-print the error message
-clear variables
-return 1
-set the infile to the in stream
-if the file can’t be opened
-print the error message
-clear the variables
-return 1

-read the private key
-if the verbose output is set
-print the modulus and private key

-decrypt the file
-clear variables and close files
