Vincent Siu
Vsiu2
10/27/22
Asgn5

Initial Design

Assignment 5

- randstate.c

```
#Will contain all the initialization functions for the gmp library
Declare a variable of type gmp_randstate_t, called state.
Declare an initialization function with parameter being the seed
    Call gmp_randinit_mt function with the parameter being state.
    #will initialize state with mersenne twister algorithm.

    Call gmp_randseed_ui function with parameter being state and the other
being seed.

#we also need to have a function that will clear the memory
Declare a clear function taking no arguments
    Call gmp_randclear and specify the state to clear.
```

- Numtheory.c

```
#will contain all the math implementations that will be used in certain calculations
for RSA keys.

#implement a power mod function
Declare the function with 4 parameters, being out, a,d,n
Set out = 1
Set a mpz variable p equal to a.
While d is bigger than 0
    If d modulo 2 equals 0
        Set out equal the product of ( out and p ) modulo n
    Set p = (p * p) mod n
    Set d = floor division of d/2
#no need to return since everything is automatically passed by pointers


#implement the greatest common denominator function
Declare the gcd function of type void taking in three arguments: d,a,b
Set d = a
While b does not equal 0:
    Set a variable temp = b
    Set b = d modulo b
    Set d equal to the temp variable
```

Vincent Siu
Vsiu2
10/27/22
Asgn5

```
#implement the inverse for modulo function
Declare the mod_inverse function of type void taking 3 parameters: i,a,n
    Set a variable r = modulo divisor that being n.
    Set a variable r_prime = the dividend, that being a.
    Set a variable q
    Set a variable temp
    Set 2 variables t and t_prime equal to 0 and 1 respectively. Use temp to aid with the var assignments.
    While r_prime is not equal 0.
        Set q = the floor division of r/r_prime
        Set temp = r
        Set r = r_prime
        Set r prime = r-(q * r_prime)
        Set temp = t_prime
        Set t prime = t-(q*t_prime)
    If r is bigger than 1:
        Set i = 0
    If t is smaller than 0
        Set i = t + n
Declare the is_prime function of type boolean which should take in 2 parameters (n being the number to test, and k being the number of iterations)
#this function will test if some arbitrarily large number, n is prime or not and will return T or F based on that.
Set a variable to track the number of times division by 2 has happened, I will name it s.
Set a variable r = n
While r mod 2 is equal to 0:
    Divide r by 2
    Increment s by 1
Set a variable j = 1
For each iterator i starting from 1 and ending at k
    Call a random generation function ranging numbers from 2 to n - 2
    Set out = Call power mod with the parameters being (out, a,d,n)
    If out does not equal 1 and out does not equal n -1
        Set variable j = 1
        While j is smaller than or equal to s - 1 and y does not equal n - 1
            Call power_mod #the variable out will change accordingly since it is PASSED BY REFERENCE
            If out is equal to 1
                Return false
            Increment j by 1
            If out does not equal to n -1
                Return false
Return true

Declare a function called make prime. It will not return anything and takes 3 parameters: p, bits, and iterations
Generate a random number while its not up to the specified number of bits
    While the generated number modulo 2 is not equal 0
        Call the primality test and if it returns true, break.
```

- RSA Library C file

```
#implement a function to make a RSA public key
Declare a function rsa_make_pub that will not return anything and include the arguments: (prime p, prime q, product of p and q being n, public exponent e, the number of bits being n bits, and number of iterations to specify being iters.
    N_bits = the number of bits to allocate for p by generating a random number in the range [nbits/4,(3*nbits/4] by using random().
    P_bits = the original number of bits - N_bits
#Calculate the lcm for p-1 and q-1 by doing the following
gcd(out, p-1,q-1) #remember that the variable out will change accordingly.
Set out = ((p-1)*(q-1))/out
While true
    Set a variable equal to the call of mpz_urandomb with the arguments being the rop, state, and the number of bits specified.
    #This function does not return anything!
    Call gcd with the arguments being rop and lcd.
    If the number is coprime (figure out how to calculate it) with the lcm, then the result to e
    Break the loop

#implement a function to write an rsa key to a pub file.
Declare the function rsa_write_pub which will not return anything. The parameters will be n,e,s,char username, and the tile type and file name.
    Print the key onto a file with (file to print on,...Hex format specifiers…).

#implement a function to read an rsa key from a pub file.
Declare the function rsa_read_pub which will not return anything. The parameters will be n,e,s,char username, and the tile type and file name.
    Scan the key onto a file with (file to read from,...Hex format specifiers…).

#implement a function to create a new private key.
Declare a function rsa_make_priv with the parameters being d (the output), e (public exponent), prime p, and prime q.
    #compute the inverse of e modulo lcm of p-1 and q-1
    Call gcd (out,p-1,q-1)
```

```
        Set out = ((p-1)(q-1))/out
        Call inverse modulo (i, e, out)
        Set d = i

#implement a function to write a private rsa key to a pub file.
Declare the function rsa_write_pub which will not return anything. The
parameters will be n,d, and the file type and file name.
        Print the key onto a file with (file to print on,...Hex format specifiers…).

#implement a function to read a private rsa key to a pub file.
Declare the function rsa_write_pub which will not return anything. The
parameters will be n,d, and the file type and file name.
        Scan the key onto a file with (file to read from,...Hex format specifiers…).

#implement the rsa encryption algorithm
Declare the encrypt function with the parameter being c, m, e, n
        Call the power exponentiation function with the parameter being (out, m,
e)
        Set out = out modulo n.
        Set c = out


#implement the function for rsa encrypt file with parameter being input file, output
file,  number of bits, and the public exponent.
        Calculate the block size k, where it equals to the floor division of
(log(2)-1)/8
        Dynamically allocate memory that will hold k bytes using uint8_t *



#implement rsa sign
```

Vincent Siu
Vsiu2
10/27/22
Asgn5

- Decrypt.c

```
#This will contain the main function for the decrypt program
Set default variable for infile, stdin.
Set default variable for outfile, stdout.
Set default variable for private key file, rsa.priv

Using getopt
    If option i is included
        Set the infile name to what is specified.
    If option o is included
        Set the outfile name to what is specified.
    If option n is included
        Set the private key file to whatever specified
    If option v is enabled.
        Enable printing in cli
    If option h is enabled
        Print the program usage.
Using fopen(), read the private key file. If there is no such file or equals NULL, print an error.

Read the private key.

If the printing is enabled, print the following
    The public modulus and the private key.

Decrypt file using rsa_decrypt_file

Close the file and clear the variables.
```

- Encrypt.c

```
#this file will contain the main function for the encrypt program
Set default variable for infile, stdin.
Set default variable for outfile, stdout.
Set default variable for public key file, rsa.pub

Using getopt
    If option i is included
        Set the infile name to what is specified.
        If option o is included
            Set the outfile name to what is specified.
        If option n is included
            Set the public key file to whatever specified
        If option v is enabled.
            Enable printing in cli
        If option h is enabled
            Print the program usage.
```

Using fopen(), read the public key file. If there is no such file or equals NULL, print an error.

Read the public key.

If the printing is enabled, print the following

Username, signature s, public modulus, and the public exponent.

Cast the username into an mpz_t type.

Call rsa_verify, and if verification fail, exit with a non 0 code.

Encrypt file using rsa_decrypt_file

Close the file and clear the variables.

- Key Generator

```
    #this will contain the implementation needed to generate an RSA key.
    Set default to variable for number of miller rabin iterations.
    Set default variable for public key file, rsa.pub
    Set default variable for private key file, rsa.priv
    Using getopt to parse the following options
        If b option is included, specify the minimum number of bits needed for public mod n
        If i option is included, specify the min numbr of miller-rabin iterations to run
        If n option is included, then specify the name of the public key file

        If the d option in included, then specify the name of the private key file.

        If the v option is included, then enable printing
```

If the h option in included, then print the program usage.

Using fopen, open both of the files specified. Print an error if this fails or file = NULL.

Set file permissions to 0600 using fchmod() and fileno()

Initialize the random state using randstate_init

Call rsa_make_pub and rsa_make_priv to make the keys.

Get the current user's name (as a string). Use getenv()

Convert the username to an an mpz_t type. Use mpz_set_str() setting the base as 62.

Call rsa_sign to compute the signature of the username.

Write the public and private keys to their respective files.

If the printing is enabled, print the following in order
- Username
- The signature s
- First large prime p
- The second large prime q
- The public modulus n
- The public exponent e
- The private key

Clear the state and clear any mpz variables made.

Vincent Siu
Vsiu2
10/27/22
Asgn5