Alex Clanton

11/9/2021

Prof.Long

CSE13S

Pseudocode/DESIGN.pdf Final

Rundown:

Make an RSA encryption program, have 3 programs where a public and private key pair is generated. An encryption program where a file is encrypted, and a decrypt program where a file is decrypted using the RSA keys you made. To facilitate the use of some of the math behind RSA, a GNU multiple-precision library will be used, in this case, GMP and PKG.

Files-

Decrypt.c

Encrypt.c

Keygen.c

Numtheory.c

Randstad.c

Rsa.c

Keygen-

Use getopt() for -b for the bits public modulus from GMP

Create input and output file pointers

Use a boolean array and optarg to store values of the selected input

-c number of iterations for Miller-Rabin prime number test

-n store the public key file

-d store private key file

-s give random seed from user

-v verbose print

-h help message

Open the files given by the user using fopen() or using the default of rsa.pub and rsa.priv

Get the int values from filno() and set permissions of 0600 so the user can only modify using fchmod()

Set seed from user -i value or default of time null

Get the user name from the environment using getenv of USER

Translate the name into a mpz_t using mpz_set_str of base 62

Call rsa_make_pub() and rsa_make_priv() passing mpz_t values

Then call rsa_sign() giving the usr mpz_t and n to create signature s

Write the public information calling rsa_write_pub() pass n,e,s and username, and file

Write the private information calling rsa_write_priv() pass n,d, and outfile

If the bool array for the selected input is true for -v print the values in base 2 for bits and the mpz_t itself

Close files and clear mpz_t

Encryptor-

Create a boolean array for the selected choices and the file variables for input file and output file

Use getopt() for -i input file, -o output of encrypted file

-n where the public key is, -v verbose, -h help message

If the user did not give default values use stdin and stdout and default to rsa.pub

Create a character array for the username

Read in the data from the given file, using rsa_read_pub() passing mpz_t values to get n,e,s, and username

Remove the newline from the username using strcspn()

Change the username into base 62 mpz_t using mpz_set_str()

Verify the signature by using rsa_verify()

If the bool array for the selected input is true for -v print the values in base 2 for bits and the mpz_t itself

Fclose the 3 files and clear the mpz's

decryptor-

Create a boolean array for the selected choices and the file variables for input file and output file

Use getopt() for -i input file, -o output of encrypted file

-n where the public key is, -v verbose, -h help message

Create mpz_t's for n and d

If the user did not give default values use stdin and stdout and default to rsa.priv

Read in the private n and d using rsa_read_priv()

Pass those values and the given input to rsa_decrypt_file()

Close the files and clear the mpz_ts

numtheory-

void pow_mod(mpz_t out, mpz_t base, mpz_t exponent, mpz_t modulus)

Create mpz_t's of v,p, val, dhold

V gets 1

P gets base

While exponent is greater than 0

    If its odd exponent

        then v gets (v*p) mod modulus

    P gets p times itself mod modulus n

    D gets d/2 (floor)

Mpz_t out gets the value

void is_prime(mpz_t n, uint64_t iters)

Create mpz_t's holding variables of s,r,temp variables

Using while loops of n-1 while mod 2 is 0

    divide of r by 2

    Add s until mod 2 is not 0

You have your r and s now

If 0-3 is the n had code for true and false and return

Create mpz_t's of j,y,a,s hold, and temp

For x=1, to k

    Random value from 2 to n-2

    y=powmod(random_val,r,n)

    If y does not equal 1 and y does not equal n-1

        J gets 1

        While j is less than equal to s-1 and y does not equal n-1

        Y gets powmod(y,2,n)

        If y equals 1

            Return false

Y does not equal n-1

Return false

Return true

void make_prime(mpz_t p, uint64_t bits, uint64_t iters)

Make mpz_t of prime and a bool value

Call mpz_urandomb giving its bit size+1 and the global state

While the number is not found prime

Check if its 2, then return

Otherwise, check if its odd then call is_prime()

If is_prime() returns false then call mpz_urandomb again do the while loop

Pass the prime number into mpz_t p

void gcd(mpz_t d, mpz_t a, mpz_t b)

Create an mpz_t t

While b is not 0

T gets b

B gets a mod b

A gets t

Set the value to d

void mod_inverse(mpz_t i, mpz_t a, mpz_t n)

Create the mpz_t's of r,r',t,t',q1,temp and temp2

R and r1 get n and a

T and t1 get 0 and 1

While r1 is not 0

Q gets r/r1 (floor)

R and r1 get r1 and r-q*r1

T and t1 get t1 and t-q*t2

If r is greater than 1

Set I to 0, no inverse

If t is less than 0

Increase t by n

Pass the t back through i

Rsa library

Bool logz2 and uint64_t logz22

User made function that computes log2 of a mpz_t

While n is greater than 0

Floor division of n by 2

Increase count by 1

If it matches nbits then return true, otherwise false

If uint64_t one return count

void rsa_make_pub(mpz_t p, mpz_t q, mpz_t n, mpz_t e, uint64_t nbits, uint64_t iters)

Create the lower and upper bounds of nbits by n/4 for lower and 3n/4 for upper

Call srand() for a random value, and mod it by (upper-lower+1) +lower

Store the values for p and q bits that add up to nbits

In a while, loop check if log n is greater than nbits

Inside the while loop call make_prime() passing p and then call it again for q

Multiply them together for n

Then find the totient by doing p-1 and q-1, multiple together

Do

Then to find public exponent take nbits and find an e calling mpz_urandomb()

Then call gcd() and see if e and the totients gcd is 1, exit while loop if it is

void rsa_write_pub(mpz_t n, mpz_t e, mpz_t s, char username[], FILE *pbfile)

Use gmp_fprintf() for the file stream of pbfile, printing n,e,s. Adding a new line each time

Then use fwrite() for the username string

void rsa_read_pub(mpz_t n, mpz_t e, mpz_t s, char username[], FILE *pbfile)

Create a holding array of characters for the user name

Use gmp_fscanf() to read in n,e,s.

Then use fread() for a sized 256 char for the username

Copy that and remove the newline

void rsa_write_priv(mpz_t n, mpz_t d, FILE *pvfile)

Use gmp_fprintf() for n and d and a newline each time

void rsa_read_priv(mpz_t n, mpz_t d, FILE *pvfile)

Use gmp_fscanf() for n and d mpz_t's values

void rsa_encrypt(mpz_t c, mpz_t m, mpz_t e, mpz_t n)

Every value in a given ciphertext c is encoded via E(m)=m^e(mod n). Use pow_mod()

void rsa_encrypt_file(FILE *infile, FILE *outfile, mpz_t n, mpz_t e)

Use the n sized data and find the block size by k floor(log2(n)-1)/9)

Use that size for a heap dynamic array size k

At 0th array element set 0 0xFF

While loop(of j receives the total bytes of fread() where the buffer arr+1 gets the data from infile)

Use mpz_import() convert read bytes with the parameters of 1 for first 1 for endian, 0 for nails. The mpz_t m takes the chunk from the array buffer.

rsa_encypt() the imported m value

Then gmp_printf() the returned ciphertext c to outfile

Free the dynamic array and clear the mpz's

void rsa_make_priv(mpz_t d, mpz_t e, mpz_t p, mpz_t q)

Create holding mpz_t's of hold,hold2,hold3

Find the totient of given p and q

Then multiple those (pq) and use mod_inverse() of that value and d,e

Set d to the out value of mod_inverse()

Clear the mpz_t's

void rsa_decrypt(mpz_t m, mpz_t c, mpz_t d, mpz_t n)

Take the message block, and run through m=c^d(mod n). Use pow_mod()

void rsa_decrypt_file(FILE *infile, FILE *outfile, mpz_t n, mpz_t d)

Create mpz_t's

Create a dynamic uint64_t for the size of exported

Use the n sized data and find the block size by k floor(log2(n)-1)/9)

Use that size for a heap dynamic array size k

At 0th array element set 0 0xFF

Use a while loop while gmp_fscanf() has not reached EOF

Call rsa_decrypt() passing the crypted c into it

mpz_export() passing the buffer and decrypted text m, the size is kept by j

fwrite() the array+1 and characters of j-1 size to the outfile

Free the buffer and j

Clear the mpz_t's

void rsa_sign(mpz_t s, mpz_t m, mpz_t d, mpz_t n)

Sign s by using s=m^d(mod n). Use pow_mod()

S is set to that value

bool rsa_verify(mpz_t m, mpz_t s, mpz_t e, mpz_t n)

Create the mpz_t hold

T gets s^e(mod n). Use pow_mod()

If t is equal to m then return true

Otherwise return false