DESIGN.pdf
Assignment 5: Public Key Cryptography

CSE 13s Winter 2022

Darrell Long

By Santosh Shrestha

Description:

This program creates an RSA public and private key which will be used to encrypt and decrypt files. The public key is used by the encryption program to create an encrypted file, whereas the private key will be used by the decryption program to decrypt and file encrypted by the encryption program.

Files to be included in directory “asgn5”:

1. decrypt.c: This contains the implementation and main() function for the decrypt program.
2. encrypt.c: This contains the implementation and main() function for the encrypt program.
3. keygen.c: This contains the implementation and main() function for the keygen program.
4. numtheory.c: This contains the implementations of the number theory functions.
5. numtheory.h: This specifies the interface for the number theory functions.
6. randstate.c: This contains the implementation of the random state interface for the RSA library and number theory functions.
7. randstate.h: This specifies the interface for initializing and clearing the random state.
8. rsa.c: This contains the implementation of the RSA library.
9. rsa.h: This specifies the interface for the RSA library.

Randstate.c

Pseudocode:

Initializing global variable

Functionality:

Initializes a global variable

Pseudocode:

Call extern with gmp_randstate_t state to create a global variable

void randstate_init

Functionality:

Initialize the random function with a seed

Pseudocode:

Initialize randstate_init as a void function that takes in an unsigned 64-bit integer(seed)
Call the function gmp_randinit_mt() with the global variable
Call gmp_randseed_ui() to set the global variable state with the value of seed
Call srandom() with the seed

void randstate_clear

Functionality:
Clears the global variable
Pseudocode:
Initialize randstate_clear as a void function that takes in a void function
Call the function gmp_randclear() with the parameter of the given global variable state
Return statement

numtheory.c

Pseudocode:

Gcd

Functionality:
Find the gcd between the 2nd and 3rd parameter and send the value of the gcd to the 1st parameter

Pseudocode:
Initialize the function gcd as a void that takes three mpz_t variables named d, a, and b
Initialize the variables aa, bb, t with the use of mpz_t and mpz_inits
Equate the variable t to 0
Equate the variable aa to the value of a
Equate the variable bb to the value of b
While loop that at iterates as long as bb is less than 0
Set the variable t to the value bb
Set the variable bb to the modulus of aa and bb
Set the variable aa to the value t
Set the variable d to the value of aa
Clear the variables aa, bb, and t

void mod_inverse

Functionality:
Finds the modular inverse of the 2nd and 3rd parameter variable and send that value to the 1st parameter variable

Pseudocode:
Initialize the function mod_inverse() as a void with the parameters of three mpz_t variables named i, a, and n
    Initialize the variables ra, rn, tzero, tone, q, holder, div with mpz_t and mpz_inits
    Set the variable ra with the value of a
    Set the variable rn with the value of n
    Set the variables div, holder, tzero, and q with the value of a 0
    Set the variable tone with the value of a 1
    While ra is not equal to 0
        Equate div to the floor division of rn divided by ra
        Equate q to div
        Equate holder to rn
        Equate rn to ra
        Equate holder to holder minus q times ra
        Equate ra to holder
        Equate holder to tzero
        Equate tzero to tone
        Equate tone to holder minus q times tone
    If rn is less than 1
        Equate i to 0
        Clear the values of ra, rn, tzero, tone, q, holder, div with mpz_clears
        Return
    If tzero is greater than 0
        Equate tzero to tzero plus n
    Equate i to tzero
    Clear the values of ra, rn, tzero, tone, q, holder, div with mpz_clears
    Return

Pow_mod

Functionality:
    Find the power modular which is the base(2nd parameter variable) to the power of the exponent(3 parameter variable) then modulus by the 4th parameter. This value is then directed to the 1st variable

Pseudocode:
Initialize the function pow_mod as a void with three mpz_t out variables named out, base, exponent, and modulus
    Initialize the variables v, bass, expo, mode with mpz_t and mpz_inits
    Equate the variable v with the value of 1
    Equate variable bass with the value of base

Equate variable expo with the value of exponent
Equate variable mode with the value of modulus
While loop that iterates as long as expo is greater than 0
If expo is odd
Equate v to the value of (v times bass) modulus mode
Equate bass to the value of (bass times bass) modulus the mode variable
Equate exponent to exponent divided by 2
Equate out to the value of v
Return

bool is_prime

Functionality:

The function finds out whether the input mpz_t variable is a prime integer and if so it returns true otherwise it will return false.

Pseudocode:

Initialize the function is_prime as a bool with the parameters mpz_t n variable(n), and an unsigned 64 bit integer variable called iters
Initialize the variables i,r, s, k, copyn, nthree, ntwo, none, y, a, j, comp with mpz_t and mpz_init
Equate the variables i, r, s, j, k, a, and y to 0
Equate the variable comp to 2
Equate the variable none to n minus 1
Equate the variable ntwo to n minus 2
Equate the variable nthree to n minus 3
Equate r to n minus 1
If n is equal to 0, 1, 2, or 3
Clear the variables i,r, s, k, copyn, nthree, none, ntwo, y, a, j, and comp with mpz_clears
Return false
While loop that iterates as long as r is even
Increment s by 1
Equate r to r divided by comp
Equate s to s minus 1
While i is less than k
Call the function mpz_urandomm with the parameters a, state, and n three
Equate variable a to a plus 2
Call function pow_mod with the parameters y, a, r, and copyn
If y is not equal to 1 and y is not equal to n minus 1
Equate j to 1
While j is less than or equals to s and y is not equal to none
Call function pow_mod with the parameters of y, y, comp, and n

If y is equal to 1
Clear the variables i,r, s, k, copyn, nthree, none, ntwo, y, a, j, and comp with mpz_clears
Return false
Equate j to j plus 1
If y is not equal to none
Clear the variables i,r, s, k, copyn, nthree, none, ntwo, y, a, j, and comp with mpz_clears
Return false
Equate i to i plus 1
Clear the variables i,r, s, k, copyn, nthree, none, ntwo, y, a, j, and comp with mpz_clears
Return true

Make_prime

Functionality:

This function creates a prime integer that is at least or above nbits long. The value is confirmed as a prime integer with is_prime.

Pseudocode:

Initialize make_prime as a void that takes in the parameters of an mpz_t variable called p, an unsigned 64-bit integer called bits, and an unsigned 64-bit integer with the variable named iters
Initialize mpz variables two, num, and byte with the mpz_t function and the mpz_inits function
Set the value of variable two with the value of 2
Set the variable num to 0
Set the variable byte to 0
Set the variable byte to the value of variable two to the power of variable bits with mpz_pow_ui
Create a boolean variable called barrier and set it to false
While loop that iterates as long as barrier is false
Set the variable num to a value from 0 to (2 to the power of bits) minus 1 with the function Mpz_urandomb
Equate num to num plus byte
If the function is_prime with the parameters of num and iters returns true
Set the value of barrier to true
Set the value of p to num
Clear all the mpz variables that were initialized in the beginning two, num, and byte
Return statement

Rsa.c

Pseudocode:

Lcm

Functionality:

Find the Lcm between the 2nd and 3rd parameter and send the lcm value to the 1st parameter variable

Pseudocode:

Initialize the function lcm as a void that takes in the parameters of three mpz_t variables called out, aval and, bval

Initialize the mpz variables copya, copyb, gcdab, and gcdval1 with the functions mpz_t and Mpz_inits

Set the variable copya to the value of aval

Set the variable copyb to the value of bval

Set copya to the absolute value of copya

Set copyb to the absolute value of copyb

Set the value of gcdab to copya times copyb

Equate gcdval1 to the gcd of copya and copyb with the gcd() function

Equate gcdval1 to be gcdab divided by gcdval1 using floor division

Set variable out to be the value of gcdval1

Clear all the mpz variables created for this function copya, copyb, gcdab, and gcdval1

Rsa_make_pub

Functionality:

The function creates a prime integer with a random amount of bits from nbits. The remaining bits are then used to create another prime integer. These prime numbers are then multiplied to create a public key which is then used for encryption. It as well creates a public exponent

Pseudocode

Initialize the function rsa_make_pub as a void that takes in the mpz_t variables p, q, n, e and the uint64_t variables nbits and iters

Initialize the mpz variables pnum, qnum, nnum, ee, gcdab, gcdval, and holder with the mpz_t and mpz_inits function

Set the variables of pnum, qnum, gcdab, ee, gcdval, holder, and nnum to the value of 0

Initialize a uint64_t variable called low with the value of nbits divided by 4

Initialize a unint64_t variable called up with the value of (3 times nbits) divided by 4

Initialize a uint64_t pbits with the value of random() value from the range of low to high by calling random() and doing a modulus of (up minus low plus 1) and adding low at the end

Initialize a unit64_t variable called qbits with the value of nbits minus pbits

Call the function make_prime() with the parameters of pnum, pbits, and iters

Call the function make_prime() with the parameters of qnum, qbits, and iters

Equate nnum to qnum times pnum
Equate p to pnum
Equate q to qnum
Equate n to nnum
Equate pnum to pnum minus 1
Equate qnum to qnum minus 1
Call the function lcm() with the parameters of gcdval, pnum, and qnum
While loop that iterates as long as ee doesn't equal 1
Equate holder to a random integer from 0 to 2^nbits -1 using the function mpz_urandomb()
Call the function gcd() with the parameters of ee, holder, and gcdval
Equate variable e to holder
Clear all the mpz variables made inside the function pnum, qnum, nnum, ee, gcdab, gcdval, and holder
return

Rsa_write_pub

Functionality:

A function that writes the public exponent, username, public key, and the two prime numbers used to create the public key into the pbfile as hex values

Pseudocode:

Initialize the function rsa_write_pub as a void that takes in the mpz_t variables n, e, and s, a char variable called username[], and a FILE called *pbfile
Call the function gmp_fprintf() that writes the variable n as a hex with a new line into pbfile
Call the function gmp_fprintf() that writes the variable e as a hex with a new line into pbfile
Call the function gmp_fprintf() that writes the variable s as a hex with a new line into pbfile
Call the function fprintf() that writes the variable username as a string with a new line into Pbfile

Rsa_read_pub

Functionality:

A function that reads the public exponent, username, public key, and the two prime numbers used to create the public key from the pbfile as hex values

Pseudocode:

Initialize the function rsa_read_pub as a void that takes in the mpz_t variables n, e, and s, a char variable called username[], and a FILE called *pbfile
Call the function gmp_fscanf() that reads the variable n as a hex with a new line from pbfile
Call the function gmp_fscanf() that reads the variable e as a hex with a new line from pbfile
Call the function gmp_fscanf() that reads the variable s as a hex with a new line from pbfile

Call the function fscanf() that reads the variable username as a string with a new line from Pbfile

Rsa_make_priv

Functionality:

A function that creates the private key for decryption

Pseudocode:

Initialize the function as a void that takes in the parameters mpz_t d, mpz_t e, mpz_t p, and mpz_t q

Initialize the mpz variables pnum, qnum, dnum, ee, gcdab, and gcdval with the mpz_t and mpz_inits function
Set the variables of pnum, qnum, gcdab, ee, gcdval, and dnum to the value of 0
Equate pnum to p
Equate qnum to q
Equate ee to e
Equate pnum to pnum minus 1
Equate qnum to qnum minus 1
Call the function lcm() with the parameters gcdab, pnum, and qnum
Call the function mod_inverse() with the parameters of dnum, ee, and gcdab
Equate d to the value of dnum
Clear all the mpz variables made inside the function pnum, qnum, nnum, ee, gcdab, and Gcdval using mpz_clears()

Rsa_write_priv

Functionality:

A function that writes the public key and private key into the pvfile as hex values

Pseudocode:

Initialize the function rsa_write_priv as a void that takes in the mpz_t variables n, d, and a FILE called *pvfile

Call the function gmp_fprintf() that writes the variable n as a hex with a new line into pvfile
Call the function gmp_fprintf() that writes the variable d as a hex with a new line into pvfile

Rsa_read_priv

Functionality:

A function the public key and private key into the pvfile as hex values

Pseudocode:

Initialize the function rsa_read_priv as a void that takes in the mpz_t variables n, d, and a FILE called *pvfile
Call the function gmp_fscanf() that reads the variable n as a hex with a new line from pvfile
Call the function gmp_fscanf() that reads the variable d as a hex with a new line from pvfile

Rsa_encrypt

Functionality:
A function that encrypts incoming messages

Pseudocode:
Initialize the function rsa_encrypt as a void that takes in the mpz_t variables c, m, e, and n
Call the function pow_mod() with the parameters c, m, e, and n

Rsa_encrypt_file

Functionality:
A function that encrypts an entire file 1 block at a time and reuses the same block. This is done by reading the same amount of bytes per block and ensuring that the end of the file hasn't been reached and importing it into a block into a file

Pseudocode:
Initialize the function rsa_encrypt_file as a void that takes in the parameters FILE infile, FILE outfile, and the mpz_t variables m, and d
Initialize the mpz variables c and m with mpz_t and mpz_init
Equate the variables c and m to 0
Initialize variable k as a size_t with the value of the bit length of n subtracted by 1 then divided by 8
Initialize variable j as a size_t
Initialize variable block as a uint8_t with the value returned from calloc when allocating space for k amount of variables with the size of uint8_t's
Set the 0th element of the block to 0xFF
While loop that iterates as long as it doesn't reach the end of the infile
Equate j to the value of bytes read and use fread() to place the read bytes into the 1st element in the block while only reading at most k-1 bytes from the infile
Import the entire block into the mpz variable m
Encrypt the variable m with the function rsa_encrypt with the parameters c, m, e, and n
Call the function gmp_fprintf to send the encrypted message as a hex into the outfile
Clear the mpz variables c and m
Free the variable block

Rsa_decrypt

Functionality:
A function that encrypts incoming messages

Pseudocode:
Initialize the function rsa_decrypt as a void that takes in the mpz_t variables m, c, d, and n
Call the function pow_mod() with the parameters m, c, d, and n

Rsa_decrypt_file

Functionality:
A function that decrypts an entire file 1 block at a time and reuses the same block. This is done by reading the same amount of bytes per block and ensuring that the end of the file hasn't been reached and exporting the decrypted block into a file

Pseudocode:
Initialize the function rsa_decrypt_file as a void that takes in the parameters FILE infile, FILE outfile, and the mpz_t variables m, and d
Initialize the mpz variables c and m with mpz_t and mpz_init
Equate the variables c and m to 0
Initialize variable k as a size_t with the value of the bit length of n subtracted by 1 then divided by 8
Initialize variable block as a uint8_t with the value returned from calloc when allocating space for k amount of variables with the size of uint8_t's
While loop that iterates as long as it doesn't reach the end of the infile
Scan the infile for a hex value and direct it to c
Decrypt the hex value with rsa_decrypt and the parameters m, c, d, n
Use mpz_export to Fill block[0] with word data from m
Write the 1st element of the block into the outfile
Clear the variables c and m
Free the array block

Rsa_sign

Functionality:
Creates a signature that is used for verification in encryption

Pseudocode:
Initialize the function rsa_sign as a void that takes in the parameters of four mpz_t variables called s, m, d, and n
Call the function pow_mod() with the parameters s, m, d, and n
Return function

Rsa_verify

Functionality:

A boolean statement that either returns true or false depending on if the signature is correct or not

Pseudocode:

Initialize the function rsa_sign as a void that takes in the parameters of four mpz_t variables called m, s, e, and n

Initialize the mpz variable snum with the functions mpz_t and mpz_init
Set the value of snum to 0
Call the function pow_mod() with the parameters snum, s, e, and n
If snum is not equal to m
Return false
Clear the variable snum
Return true

keygen.c

Pseudocode:

Usage

Initialize usage as a void and its parameters as a void

Call fprintf stderr and write about the function and how to use it and call exec at the end of the message

Main

Initialize main as an int that takes in the parameters of an int argc and a char **argv

Initialize seed as a uint64_t with the value of returned from function time() with the parameters of NULL
Call srandom() with parameter seed
Initialize user as a char with the value returned from using getenv() with the parameter “USER”
Initialize bits as a uint64_t with the value of 256
Initialize confidence as a uint64_t with the value of 50
Initialize verbose as a bool with the value of false
Initialize opt as an int with the value of 0
Initialize a char named pvstring with the string rsa.priv
Initialize a char name pubstring with the string rsa.pub
While loop that set opt equals to getopt(argc, argv, "h, v, c:, b:, n:, d:, s:")) != -1))

Call switch function with the parameter of opt
Case “h”
Call function usage() with parameter of argv[0]
Return EXIT_FAILURE
Case “v”
Equate verbose to true
Break statement
Case “b”
Equate bits to the user input converted from string to uint64_t
Break statement
Case “c”
Equate confidence to the user input converted from string to uint64_t
Break statement
Case “n”
Equate pubstring to optarg
Break statement
Case “d”
Equate pvstring to optarg
Break statement
Case “s”
Equate seed to the user input converted from string to uint64_t
Break statement
Set the default case to direct the user to the user page
Return EXIT_FAILURE
Initialize pvfile as a FILE pointer with a fopen() to pubstring to write
If pvfile is equal to NULL
Print the message "Error opening *filename*”
return EXIT_FAILURE
Initialize pubfile as a FILE pointer with a fopen() to pvstring to write
If pubfile is equal to NULL
Print the message "Error opening *filename*”
return EXIT_FAILURE
Initialize the mpz variables n, d, e, q, p, s, m with the functions mpz_t and mpz_inits
Call randstate_init() with the parameter of seed
Initialize pvnumber with the value returned from fileno() with the parameter of pvfile
Call fchmod() with the parameter of pvnumber and 0600
Call rsa_make_pub with the parameters p, q, n, e, bits, and confidence
Call rsa_make_priv with the parameters d, e, p, q
Convert user to a string into the mpz variable m with the mpz_set_str function with a base of 62

Call rsa sign with the parameters of s, m, d, and n
Call the function rsa_write_pub with the parameters of n, e, s, user and pubfile
Call the function rsa_write_priv with the parameters of n, d, user, and pvfile
If verbose equals true
Initialize the variable qbits as a size_t with the value of bits within q
Initialize the variable pbits as a size_t with the value of bits within p
Initialize the variable nbits as a size_t with the value of bits within n
Initialize the variable sbits as a size_t with the value of bits within s
Initialize the variable dbits as a size_t with the value of bits within d
Initialize the variable ebits as a size_t with the value of bits within e
Call the function printf() to print what the users username is
Call the function gmp_printf() to print the number of bits within variable s
Call the function gmp_printf() to print the number of bits within variable p
Call the function gmp_printf() to print the number of bits within variable q
Call the function gmp_printf() to print the number of bits within variable n
Call the function gmp_printf() to print the number of bits within variable e
Call the function gmp_printf() to print the number of bits within variable d
Call fclose for pvfile
Call fclose for pubfile
Call randstate_clear()
Clear the mpz variables n, d, e, q, p, s, and m
Return 0

Encrypt.c

Pseudocode:

Usage

Initialize usage as a void and its parameters *exec as a char
Call fprintf stderr and write about the function and how to use it and call exec at the end of the message

Main

Initialize main as an int that takes in the parameters of an int argc and a char **argv
Initialize user as a char with 256 allocated space with the value of {0}
Initialize verbose as a bool with the value of false
Initialize opt as an int with the value of 0
Initialize infile as a FILE pointer as stdin
Initialize outfile as a FILE pointer as stdout
Initialize a char name pubman with the string rsa.pub
While loop that set opt equals to getopt(argc, argv, "h, v, i:, o:, n:")) != -1))

Call switch function with the parameter of opt
Case “h”
Call function usage() with parameter of argv[0]
Return EXIT_FAILURE
Case “v”
Equate verbose to true
Break statement
Case “i”
Equate infile to an fopen function call with the user input as a file to read from
If infile is equal to NULL
Print the message "Error opening *filename*”
return EXIT_FAILURE
Break statement
Case “n”
Equate pubman to optarg
Break statement
Case “o”
Equate outfile to an fopen function call with the user input as a file to read from
If outfile is equal to NULL
Print the message "Error opening *filename*”
return EXIT_FAILURE
Break statement
Set the default case to direct the user to the user page
Return EXIT_FAILURE
Initialize pubfile as a FILE pointer with a fopen() to pubman to read
If pubfile is equal to NULL
Print the message "Error opening *filename*”
return EXIT_FAILURE
Initialize the mpz variables n, e, s, d, m with the functions mpz_t and mpz_inits
Call rsa_read_pub with the parameters of m, n, e, s, user, and pubfile
If verbose is equal to true
Initialize the variable nbits as a size_t with the value of bits within n
Initialize the variable sbits as a size_t with the value of bits within s
Initialize the variable ebits as a size_t with the value of bits within e
Call the function printf() to print what the user's username is
Call the function gmp_printf() to print the number of bits within variable s
Call the function gmp_printf() to print the number of bits within variable n
Call the function gmp_printf() to print the number of bits within variable e
Convert user to a string into the mpz variable m with the mpz_set_str function with a base of 62

If rsa_verify() with the parameters of m, s, e and n return false
Print the message "Error signature could not be verified”
return EXIT_FAILURE
Call the function rsa_encrypt_file with the inputs of infile, outfile, n, and e
Clear the mpz variables m, n, e, s, and d
Call fclose for infile
Call fclose for outfile
Call fclose for pubfile
Return 0

decrypt.c

Pseudocode:

Usage

Initialize usage as a void and its parameters *exec as a char
Call fprintf stderr and write about the function and how to use it and call exec at the end of the Message

Main

Initialize main as an int that takes in the parameters of an int argc and a char **argv
Initialize verbose as a bool with the value of false
Initialize opt as an int with the value of 0
Initialize infile as a FILE pointer as stdin
Initialize outfile as a FILE pointer as stdout
Initialize pvstring as a char with the string rsa.priv
While loop that set opt equals to getopt(argc, argv, "h, v, i:, o:, n:")) != -1))
Call switch function with the parameter of opt
Case “h”
Call function usage() with parameter of argv[0]
Return EXIT_FAILURE
Case “v”
Equate verbose to true
Break statement
Case “i”
Equate infile to an fopen function call with the user input as a file to read from
If infile is equal to NULL
Print the message "Error opening *filename*”
return EXIT_FAILURE
Break statement

Case “n”
Equate pvstring to optarg
Break statement
Case “o”
Equate outfile to an fopen function call with the user input as a file to read from
If outfile is equal to NULL
Print the message "Error opening *filename*”
return EXIT_FAILURE
Break statement
Set the default case to direct the user to the user page
Return EXIT_FAILURE
Initialize pvfile as a FILE pointer with a fopen() to pvstring to read
If pvfile is equal to NULL
Print the message "Error opening *filename*”
return EXIT_FAILURE
Initialize the mpz variables n and d with the functions mpz_t and mpz_inits
Call rsa_read_priv with the parameters of n and d, and pvbfile
If verbose is equal to true
Initialize the variable nbits as a size_t with the value of bits within n
Initialize the variable dbits as a size_t with the value of bits within d
Call the function gmp_printf() to print the number of bits within variable n
Call the function gmp_printf() to print the number of bits within variable d
Call the function rsa_decrypt_file with the parameters of infile, outfile, n, and, d
Call the function fclose() for pvfile
Call the function fclose() for infile
Call the function fclose() for outfile
Clear the variables mpz n and d
Return 0

Credits

- I got the implementation of the lcm function, how to create create a lower and upper bound for the random value, implementation of lambda from Eugene's section
- I got the idea of using feof() from Brian and he helped me understand the syntax needed for the rsa encrypt and decrypt file functions
- I got all my usage pages from the usage pages of the resources code given
- I used the pseudocode given for the numtheory.c functions
- I got the idea of how to use char user[256] from Omar