Nicolas Corfmat
CSE13S, Winter 2023
ncorfmat@ucsc.edu

Assignment 5 DESIGN.pdf

1 Description of Program:

This project consists of three main programs: *keygen*, *encrypt*, and *decrypt*. The *keygen* program generates SS public-private key pairs which are then used by *encrypt* and *decrypt*. The public key is used by *encrypt* to securely encrypt files, while *decrypt* makes use of the private key to unlock, or decrypt, files.

2 Files to be included in directory “asgn5”:

1. decrypt.c:
    - Contains the main() function for the decrypt program.
2. encrypt.c:
    - Contains the main() function for the encrypt program.
3. keygen.c:
    - Contains the main() function for the keygen program.
4. numtheory.c
    - C file containing the number theory functions.
5. numtheory.h
    - Header file containing the interface to numtheory.c
6. randstate.c
    - C file containing the implementation of the random state interface.
7. randstate.h
    - Header file containing the interface to randstate.c
8. ss.c
    - C file containing the implementation of the SS library.
9. ss.h
    - Header file containing the interface to ss.c
10. Makefile:
    - Supports the compilation of the all .c programs in the “asgn5” directory.

11. README.md:
    - Proper explanation of how to run the program and *Makefile*. It also contains a list of available command-line options and explains their functions, as well as any false positives reported by scan-build.
12. DESIGN.pdf:
    - Explains the thought process behind the program with sufficient detail. Comes in PDF format.
13. WRITEUP.pdf:
    - PDF file created using LaTeX that explains the applications of public-private cryptography and describes what I learned from this assignment.

3 Pseudocode / Structure:

randstate.c

Create a global random variable

Create a function that initializes the random variable and takes a seed as input:
    Generate a random number from the seed

    Initialize the random variable
    Set the random variable to the random number generated from seed

Create a function that clears the random variable:
    Clear the random variable

numtheory.c

Create a function that returns the pow mod of a number:
    Assign the number 1 to ‘out’

    While the exponent is greater than 0:
        If the exponent is odd:
            Multiply (out by base) and mod the result by the given modulus
        Multiply the base by itself and mod the result by the given modulus
        Divide the exponent by 2 (floor division) and overwrite the current base

    Set ‘out’ to calculated value
    Deallocate memory

Create a function that return true or false depending on whether the given number is prime:

If the number is less than or equal to 1:

Number is not prime

If the number is either 2, 3, or 5:

Number is prime

If the number is even:

Number is not prime

Subtract 1 from given number and assign value to ‘r’

While ‘r’ is not odd:

Increment ‘s’ (starting from 0)

Increment ‘r’

Subtract 1 from ‘s’

For every number up to ‘iterations’ beginning with 1:

Choose a random element from the set {2, 3, …, n - 2}

Pow mod using the element as the base, ‘r’ as the exponent, and the passed number argument as the modulus, and assign the result to ‘y’

If y is not equal to 1 and n - 1:

Assign 1 to j

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

Update y with a new Pow Mod, this time with the arguments y, 2, and the potential prime number

If y is 1:

Deallocate memory and return false, indicating number is not prime

Add 1 to j

If y is not equal to n - 1

Deallocate memory and return false, indicating number is not prime

Deallocate memory and return true, indicating that the number is prime

Create a function that makes a prime number with X number of bits:

Assign 0 to ‘out’

While out is not prime:

Generate a new random number consisting of X bits and set it to out

Create a function that calculates the greatest common divisor of two numbers, a and b:

While b is not equal to 0:

Assign b to a temporary variable

Calculate a mod b and overwrite b with the result

Assign the temporary variable to a

Set ‘d’ to greatest common divisor and deallocate memory

Create a function that calculates the inverse of a modulus:

Store the modulus in a new variable, r

Store the inverse in a new variable, r’

Assign 0 to a new variable, t

Assign 1 to a variable, t’

While r’ is not equal to 0:

Divide r by r’ and store the result in a new variable, q

Update r with r’ and r’ with (r - q x r’)

Update t with t’ and t’ with (t - q x t’)

If r is greater than 1:

Return no inverse

If t is less than 0:

Update t with the sum of t and the modulus

Return t, or the new modulus and deallocate memory

ss.c

Create a function that generates a public key:

Loop while sqrt(n) is greater than given bits, p is not divisible by (q - 1), and q is not divisible by (p - 1):

Compute bit counts for p and q

Compute two prime numbers, p and q with respective bit counts

Deallocate memory

Create a function that writes the public key into a file:

Write public key (as hexstring) and username to public file, each with a trailing new line

Create a new function that reads a public key from a given file:

Read public key (as hexstring) and username from public file, each with a trailing new line

Create a new function that makes a private key:

Compute $\lambda = \text{lcm}(p - 1, q - 1)$

Determine the mod inverse of private key and λ

Store the result in the given variable, d, and deallocate memory

Create a new function that writes the private key into a given file:

Write pq (as hexstring) and d (as hexstring) to private file, each with a trailing new line

Create a new function that reads a private key from a given file:

Read pq (as hexstring) and d (as hexstring) from private file, each with a trailing new line

Create a function that performs the SS encryption:

Compute the ciphertext by encrypting the given message, using the formula:

$E(m) = c = m^n \pmod{n}$

Create a function that encrypts a file:

Calculate block size ‘k’ as k = [(log2 (sqrt(p))−1)/8]
Allocate memory for an array that holds ‘k’ blocks
Set the zeroth byte of the block to 0xFF

For every line in input file:

Read at most k−1 bytes in from input file, with j being the number of bytes actually read
Convert the read bytes
Encrypt m and write the encrypted number to output file as a hexstring followed by a trailing newline

Deallocate block and created objects

Create a function that performs the SS decryption:

Compute the ciphertext by encrypting the given message, using the formula:
$D(c) = m = c^d \pmod{pq}$

Create a function that encrypts a file:

Calculate block size ‘k’ as k = [(log2 (sqrt(p))−1)/8]
Allocate memory for an array that holds ‘k’ blocks
Set the zeroth byte of the block to 0xFF

For every line in input file:

Read the hexstring from input file and save it as ‘c’
Decrypt ‘c’ back to ‘m’
Convert ‘m’ back to bytes
Write out j−1 bytes starting from index 1 to output file

Deallocate block and created objects

keygen.c

Main function:

Initialize *bits* to false
Initialize *iters* to 256
Initialize *seed* to time since UNIX epoch (in seconds)

Initialize verbose output to false

Initialize username to NULL
Initialize public key file to “ss.pub”
Initialize private key file to “ss.priv”

While retrieving arguments from command line:

If user types b:

Specify *bits*

If user types i:

Specify *iters*

If user types n:

Takes the user-specified public key file

If user types d:

Takes the user-specified private key file

If user types s:

Specify *seed*

If user types v:

Enable verbose output

If user types h:

Display program usage

Open public and private key files, printing an error message upon failure
Set private key file permissions are set to read and write
Initialize mpz objects and random global variable ‘state’

Generate public and private keys
Retrieve the user’s name

Write public and private keys to their respective files

If verbose output is enabled:

Print username, along with mpz objects (as decimals)

Close files
Clear both random ‘state’ and mpz objects

encrypt.c

Main function:

Initialize verbose output to false

Initialize username to login name
Initialize public key file to “ss.pub”
Initialize input file to stdin
Initialize output file to stdout

While retrieving arguments from command line:
If user types i:
Specify input file
If user types o:
Specify output file
If user types n:
Specify public key file
If user types v:
Enable verbose output
If user types h:
Display program usage

Open public key file, printing an error message upon failure
Open input and output files

Initialize mpz objects

Read public key from the public key file

If verbose output is enabled:
Print username, along with mpz objects (as decimals)

Encrypt input file with given public key

Close public key file
Deallocate mpz objects

decrypt.c

Main function:

Initialize verbose output to false

Initialize private key file to “ss.priv”
Initialize input file to stdin
Initialize output file to stdout

While retrieving arguments from command line:
If user types i:
Specify input file
If user types o:
Specify output file
If user types n:
Specify private key file
If user types v:
Enable verbose output
If user types h:
Display program usage

Open private key file, printing an error message upon failure
Open input and output files

Initialize mpz objects

Read private key from the private key file

If verbose output is enabled:
Print mpz objects (as decimals)

Decrypt input file with given private key

Close private key file
Deallocate mpz objects