Assignment 5 DESIGN.pdf

Vincent Liu
2/26/23

Purpose:

The purpose of this assignment is for students to understand the process in which encryption, decryption and key generation is done. While also expanding student experience & understanding of these concepts, students will be exposed to further fundamental concepts found within Computer Science. Students are expected to create three main files for this assignment. A key generator, an encryptor, and a decryptor. Additionally students will be creating supplemental c files that contain the required functions for this assignment. These consist of ss.c (to implement the Schmidt-Samoa Algorithm), randstate.c and numtheory.c. Students will also be utilizing the GNU MP library for multi precision arithmetic integers, as C does not natively support large precision integers; A key factor in the Schmidt-Samoa cryptosystem, as it is heavily reliant on large integer factorization.

Note*
-Almost all of the pseudocode for this assignment was provided in the assignment documentation.

Files in Directory:

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 SS library and number theory functions.
7. randstate.h:
    - This specifies the interface for initializing and clearing the random state.
8. ss.c:
    - This contains the implementation of the SS library.
9. ss.h:

    - This specifies the interface for the SS library.
10. Makefile
    - Used to clean directory, generate associated executable file, and properly format .c files
11. README.md
    - Text file in Markdown format that describes how to build and run the program.
12. DESIGN.pdf
    - covers the purpose of the program
    - layout/structure of the program
13. WRITEUP.pdf
    - Describes what the program does, gives insight on the results found, explains what was learned, and clarifies steps taken to complete the assignment.

Structure of Program//Pseudocode (a):

Main files:

decrypt.c

- Use getopt to accept commands from terminal
    - -i : specifies the input file to decrypt (default: stdin).
        - If this is given, set using stdin to false
    - -o : specifies the output file to decrypt (default: stdout).
        - If this is given, set using stdout to false
    - -n : specifies the file containing the private key (default: ss.priv).
        - If this is given, set using ss_priv to false
    - -v : enables verbose output.
        - Set verbose to true
    - -h : displays program synopsis and usage.
    - If an argument is given but no following argument, print error synopsis
- Attempt to open the private key file. Prints error message if fail to openvi denc
- Opens the input file and output files (if valid arguments are passed in)
    - If input is null, print error message
- Read private key
- If verbose is true, print verbose output
- Decrypt file using ss_decrypt_file()
- Close all opened files
- Clear mpz variables

encrypt.c

- Use getopt to accept commands from terminal
    - -i : specifies the input file to encrypt (default: stdin).
        - If this is given, set using stdin to false
    - -o : specifies the output file to encrypt (default: stdout).
        - If this is given, set using stdout to false
    - -n : specifies the file containing the public key (default: ss.pub).
        - If this is given, set using ss_priv to false
    - -v : enables verbose output.
        - Set verbose to true
    - -h : displays program synopsis and usage.
    - If an argument is given but no following argument, print error synopsis
- Attempt to open the public key file. Prints error message if fail to open
- Opens the input file and output files (if valid arguments are passed in)
    - If input is null, print error message
- Gets username of user
- Read public key
- If verbose is true, print verbose output
- Encrypt file using ss_encrypt_file()
- Close all opened files
- Clear mpz variables

keygen.c

- Helper function to help print verbose output
    - Variable name, num of bits, and value
- Use getopt to accept commands from terminal
    - -b : specifies the minimum bits needed for the public modulus n.
        - Set minimum bits to specified number
    - -i : specifies the number of Miller-Rabin iterations for testing primes (default: 50).
        - Set minimum Miller-Rabin iterations to specified number
    - -n pbfile : specifies the public key file (default: ss.pub).
        - Set using_ss_pub to false
    - -d pvfile : specifies the private key file (default: ss.priv).
        - Set using_ss_priv to false
    - -s : specifies the random seed for the random state initialization (default: the seconds since the UNIX epoch, given by time(NULL)).
        - Set seed to optarg
    - -v : enables verbose output.
        - Sets verbose to true
    - -h : displays program synopsis and usage.
- If using using_ss_pub or using_ss_priv is true,

    - Public key file is ss.pub
    - Private key file is ss.priv
- Open public and private keys. Create the files if files do not already exist Print error and exit program in event of error
- Set private key permissions to be read and write for user only
- Initialize random state with randstate_init(seed)
- Make the public and private keys
- If verbose is true, print verbose output
- Close all opened files
- Clear randstate
- Clear mpz variables

Structure of Program//Pseudocode (b):

Files with associated functions:

randstate.c

- **void randstate_init(uint64_t seed)**
    - Calls srand() using given seed
    - Initializes gmp random state named “state”
    - Calls gmp_randstate_ui(state, seed)

- **void randstate_clear(void):** Clears and frees memory.
    - Uses gmp_randclear(void) instead of free() to free up allocated memory

---

numtheory.c

- **void pow_mod(mpz_t o, mpz_t a, mpz_t d, mpz_t n) : performs modular exponentiation.**
    - Initialize variables
    - Set variables
    - o=1
    - p=a
    - While d>0
        - If d is odd
            - o = (o*p) mod(n)
            - o = (o*o) mod(n)
        - p=p*p
        - p = (p*n) mod(n)
        - d=(d/2)
    - Clear mpz variables

- **bool is_prime(mpz_t n, uint64_t iters):** Checks if a specified integer is a prime number. Used provided pseudocode from assignment pdf
    - If n is even, return n=0
    - If n is 1, return false
    - If n is 3 return true
    - Initialize variables
    - Set variables
    - For i in k
        - Choose a random integer $a \in \{2, 3, \ldots, n-2\}$
        - y= pow_mod(a,r,n)
        - If y!=1 and y!=n-1
            - While j <=s-1 and y!=n-1
                - j=1
                - If y==1
                    - Return false
                - j+=1
            - If y!=n-1
                - Return false
    - Return true
- **void make_prime(mpz_t p, uint64_t bits, uint64_t iters)**
    - Initialize variables
    - Set variables
    - Generate a new number and store in p
    - Ensure it is prime and is at least ‘bits’ long
    - Primality tested using *is_prime()* using *iters* number of iterations.
    - Clear mpz variables

- **void gcd(mpz_t d, mpz_t a, mpz_t b) : finds the greatest common divisor between a and b and store into variable g**
    - Initialize variables
    - Set variables
    - While b!=0
        - t=b
        - b = a mod b
        - a = t
    - “Return” d
    - Clear mpz variables

- **void mod_inverse(mpz_t i, mpz_t a, mpz_t n) : computes the inverse of 'a mod n' and store it into mpz variable o**
    - Initialize variables
    - Set variables
    - r=n
    - r_prime=a
    - t=0
    - t_prime=1
    - While r'!=0
        - q=r/r_prime
        - r=r_prime
        - r'=(r-(q*r_prime))
        - t=t_prime
        - t_prime=(t-(q*t_prime))
    - If r>1
        - Return no inverse
    - If t<0
        - t=t+n
    - Clear mpz variables
    - Return t

---

ss.c

- **void ss_make_pub(mpz_t p, mpz_t q, mpz_t n, uint64_t nbits, uint64_t iters)**
    - Initialize mpz variables
    - Set min variable to be (nbits/5)
    - Set max variable to be ((2*nbits)/5)
    - while
        - Make primes
            - If lower than min or higher than max, restart
            - Else break
    - Compute $\lambda(n) = \text{lcm}(p-1,q-1)$ by doing $\gcd(p-1,q-1)$
    - Set n and return

- **void ss_write_pub(mpz_t n, char username[], FILE *pbfile)**
    - mpz_out_str the value of n into pbfile as a hexstring
    - Print newline
    - Print username into file
    - Print newline

- **void ss_read_pub(mpz_t n, char username[], FILE *pbfile)**
    - mpz_inp_str contents of pbfile from hexstring and place into variable n
    - Scanf for username

- **void ss_make_priv(mpz_t d, mpz_t pq, mpz_t p, mpz_t q)**
    - Creates a new private SS private key
    - gcd=((p-1)*(q-1))
    - lcm=lcm/gcd
    - d=p*p*q
    - pq=p*q

- **void ss_write_priv(mpz_t pq, mpz_t d, FILE *pvfile)**
    - mpz_inp_str contents of variables pq and d into pvfile as hexstring followed by newlines

- **void SS_read_priv(mpz_t pq, mpz_t d, FILE *pvfile)**
    - mpz_inp_str of contents from pvfile into variables pq and d

- **void ss_encrypt(mpz_t c, mpz_t m, mpz_t n)**
    - Performs ss encryption via c = m^(n)(mod n)

- **void ss_encrypt_file(FILE *infile, FILE *outfile, mpz_t n)**
    - Encrypts contents of infile (written in blocks) and writes to outfile
    - Initialize variables
    - Calculate block size k (k= | (log_2(sqrt(n)-1)/8)) |)
    - calloc() an array of type (uint32_t *) that holds k bytes
    - Set 0th byte of block to 0xFF
    - While not end of file
        - At most read k-1 bytes from infile
        - Let j be number of bytes read
        - Place read bytes into allocated block starting from intex i. DO NOT OVERWRITE 0xFF
        - mpz_import() convert read bytes into mpz_t m
        - Encrypt m with ss_encrypt(), then write to outfile as hexstring followed by newline
    - Free block
    - Clear mpz variables

- **void ss_decrypt(mpz_t m, mpz_t c, mpz_t d, mpz_t pq)**

    - Compute message m by decrypting ciphertext c using priv key d and public modulus n
    - m=c^(d) (mod pq)
        - Use pow_mod

- **void ss_decrypt_file(FILE *infile, FILE *outfile, mpz_t pq, mpz_t d)**
    - Initialize variables
    - Calculate value of k (k= | (log_2(sqrt(n)-1)/8)) |)
    - Allocate memory for array (uint32_t *) that can hold k bytes.
        - This will serve as block (variable name block2)
    - While not end of file
        - Scan and save hexstring from infile as mpz_t c
            - If c is a newline, break
        - Decrypt c back into original value m using ss_decrypt
        - Using mpz_export() convert m back into bytes and store them back into block2
        - Write contents of block2 +1, bytes k-1 to outfile
    - Clear mpz_variables
    - Free block2

Makefile

- Set compile for C language
- Cflags -Wall -Wpedantic -Werror -Wextra -standard c17
- Objects tag for ‘numtheory.o’, ‘randstate.o’, ‘ss.o’
- Generate ‘decrypt’, ‘encrypt’, and ‘keygen’ executables if “all” or “make” command is given
- Generate all .o files from .c files and associated header files
- Exclusively create ‘decrypt’, ‘encrypt’, or ‘keygen’ given the command
- Clean:
    - Removes all executable files as well as associated ‘.o’ files
- Format:
    - Formats all c files to c17 standard