Design Document

Manikanta Illuri

Initial Design

Objective: The objective of this assignment is to create a program that can encrypt and decrypt a given input. For this assignment I will be using keygen to produce an ss public and private key pair and use the encrypt and decrypt files to manipulate the input.

Deliverables:

- Keygen.c
- Encrypt.c
- Decrypt,c
- Numtheroy.c
- Numtheory.h
- Randstad.c
- Randstate.h
- Ss.c
- ss.h
- Makefile
- DESIGN.pdf
- WRITEUP.pdf
- README.md

Num Theory

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

- This function will perform modular exponentiation by computing the base raised to the exponent power modulo modulus and storing the value in an out var. I will be implementing a while loop that will check the value greater than 0. Within the loop I will use temp variables and a modulus operator to initiate the exponentiation. Outside the loop I will multiply the two temp values.

bool is_prime(mpz_t n, uint64_t iters)

- Thor function will indicate if a number is prime or not. To do so the function will have temp variables that are set to NULL. Then the variables will be sent to a while loop that checks if temp != 0. Within the loop an if will check if the values are divisible then it will use the modulus operation.

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

- This function is meant to generate prime numbers that are at least a bit length long. For this implementation I will initiate 2 temp variables that will be set to NULL. Then using a

while loop I will check if the conviction while temp ==0, then it will call random and pass in n bits as an argument. Outside the loop it will clear the variables.

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

- For the function I will be writing a program that computes the greatest common divisor of a and b and store them in temp variables. I will be using a loop that will check if temp is != 0 and will swap the values and mod them returning the other variable.

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

- This function is used to compute the inverse i of a module n. For the implementation of this function I will begin a while loop that will run while the temp variable is prime. And is not equal to 0. Within the while loop another nested loop will be present that will use the div function to divide the values of the two temp variables. Then it will return the values outside the loop.

SS Library

void ss_make_pub(mpz_t p, mpz_t q, mpz_t n, uint64_t nbits, uint64_t iters)

- For the implementation of this function I will be having two large prime numbers and will compe n by doing p*p*q. For this function I plan on using a while loop that iterates while nbits is /5 is greater than 0. Inside the I will perform the multiplication operator and will set them the temp variable to the final value.

void ss_write_pub(mpz_t n, char username[], FILE *pbfile)

- For this function I will make a function that will write to teh ss key file. I will use the fscanf function to print to the output file.

void ss_read_pub(mpz_t n, char username[], FILE *pbfile)

- For this function I will fscanf

void ss_make_priv(mpz_t d, mpz_t p, mpz_t q)

- For the make private using a set of temp variables and by setting them equal to NULL. I will also use the abs function to store them into the output file.

void ss_make_pub(mpz_t d, mpz_t p, mpz_t q)

- For the make public using a set of temp variables and by setting them equal to NULL. I will also use the abs function to store them into the output file.

void ss_write_priv(mpz_t pq, mpz_t d, FILE *pvfile)

- I will havea ss key file that I will write to using the w permission.

void SS_read_priv(mpz_t pq, mpz_t d, FILE *pvfile)

- For the function I will use the ssh key output file and read the file by using the r permissions.

void ss_encrypt(mpz_t c, mpz_t m, mpz_t n)

- This function will use calloc and a while loop to allocate memory to the temp variables. I will use a while condition that will check if it is the end of the file. Then it will encrypt the input to the output file.

void ss_encrypt_file(FILE *infile, FILE *outfile, mpz_t n)

- This function will use the contents of the infile and will encrypt them using modn and a while loop. And will check the below
- While there are still unprocessed bytes in infile: (a) Read at most k−1 bytes in from infile, and let j be the number of bytes actually read. Place the read bytes into the allocated block starting from index 1 so as to not overwrite the 0xFF. (b) Using mpz_import(), convert the read bytes, including the prepended 0xFF into an mpz_t m. You will want to set the order parameter of mpz_import() to 1 for the most significant word first, 1 for the endian parameter, and 0 for the nails parameter. (c) Encrypt m with ss_encrypt(), then write the encrypted number to outfile as a hex string followed by a trailing newline.

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

- This function will use the ss keygen and take the outfile and decrypt the message within.
- It will check based on the below.
    - Iterating over the lines in infile: (a) Scan in a hex string, saving the hexstring as a mpz_t c. Remember, each block is written as a hex string with a trailing newline when encrypting a file. (b) First decrypt c back into its original value m. Then, using mpz_export(), convert m back into bytes, storing them in the allocated block. Let j be the number of bytes actually converted. You will want to set the order parameter of mpz_export() to 1 for the most significant word first, 1 for the endian parameter, and 0 for the nails parameter. (c) Write out j−1 bytes starting from index 1 of the block to outfile. This is because index 0 must be prepended 0xFF. Do not output the 0xFF.

Key Generator

This file is the main file with the main function in it. It can take the options of:

• -b : specifies the minimum bits needed for the public modulus n.
• -i : specifies the number of Miller-Rabin iterations for testing primes (default: 50).
• -n pbfile : specifies the public key file (default: ss.pub).
• -d pvfile : specifies the private key file (default: ss.priv).

• -s : specifies the random seed for the random state initialization (default: the seconds since the UNIX epoch, given by time(NULL)).
• -v : enables verbose output.
• -h : displays program synopsis and usage.
The program will use get opt to set these flags an fopen and close to manipulate the in and out files from the other functions.

Encrypt

This file is meant to act as the main function for the encryption program. This program will have the flags of:
• -i : specifies the input file to encrypt (default: stdin).
• -o : specifies the output file to encrypt (default: stdout).
• -n : specifies the file containing the public key (default: ss.pub).
• -v : enables verbose output.
• -h : displays program synopsis and usage
The function will use ss_encrpyt_file and use the mpz_t to encrypt the file. It will take in the username and the public key n,. The program will use get opt to set these flags an fopen and close to manipulate the in and out files from the other functions.

Decryptor

Thif file will be the decrpyteor file. It will take the flash of:
• -i : specifies the input file to decrypt (default: stdin).
• -o : specifies the output file to decrypt (default: stdout).
• -n : specifies the file containing the private key (default: ss.priv).
• -v : enables verbose output.
• -h : displays program synopsis and usage.
This file will use the private modulus pq and the private key d to decrypt the message that came in through the output file of the other file.

Randstatate

Will set gmp_ranstate_t and state
void randstate_init(uint64_t seed)

- This function will use a srandom seed and generate the state seed.

Void rnadtstate_clear(void)

- Will use gmp_rnadclear and pass in state.