Assignment 5

DESIGN

Prof Veenstra

Nguyen Vu

1. numtheory.c

This first section will take about the design of the file numtheory.c

1.1. void gcd(mpz t d, mpz t a, mpz t b)

The function first initializes a temporary variable r. The function then uses Euclid's algorithm to compute the GCD of a and b. The algorithm repeatedly divides a by b, setting a to the remainder and b to the previous value of a. It continues doing this until b is zero. Once the GCD is computed, the function sets the result to d. Finally, the temporary variable r is cleared.

Pseudo Code:

```
Initialize temporary variable r
Calculate the GCD using Euclid's algorithm
Set the result to d
Clear the temporary variable
```

1.2. void mod inverse(mpz t i, mpz t a, mpz t n)

The function first initializes the necessary mpz_t variables, including temporary variables used in the extended Euclidean algorithm. It then sets up the initial values for 't', 't prime', 'r', and 'r prime' as described in the algorithm pseudo code. The function then enters a loop and executes the extended Euclidean algorithm to get 'q', 'r' and 't'. It updates the values of 't', 't prime', 'r', and 'r prime' for the next iteration of the loop. After the loop completes, the function checks whether 'r' is equal to 1. If it is, the function sets i to the value of t, which is the modular inverse of a modulo n. If r is not equal to 1, it means that a does not have a modular inverse modulo n, so the function sets i to 0. Finally, the function clears the mpz_t variables that it allocated using mpz_clears.

Pseudo Code:

```
Set r = a, r prime = n, t = 1, t prime = 0
Make a while loop for when r prime > 0 {
Calculate the modular inverse using extended Euclidean algorithm }
Check if 'r' = 1, if yes, set 'i' to 't', if not (no inverses exists), set 'i' to 0
Clear the temporary variables
```

1.3 bool is_prime(mpz_t n, uint64_t iters)

The function first checks whether the number is less than 2 or even, if true then return false. It then checks if the number is 2 or 3, if true then return true. After that the function writes n-1 = d = 2^r. The function then use the Miller-Rabin method to check whether 'n' is prime by checking a^(n-1) ≡ 1 (mod n) and a^(2^r * (n-1)) ≡ -1 (mod n) for all integer r, $0 \leq r \leq s-1$. If all conditions pass, then 'n' might be prime with a 25% chance of being wrong is 25%, so the function repeats the algorithm the iters number (default 100) to lower the chance of being wrong.

Pseudo Code:

```
Check if n is even or less than 2, if yes then return false
Check if n is 2 or 3, if yes then return true
Make and initialize variables for n-1 = d = 2^r
Execute Rabin Miller algorithms repeatedly based on iters
Clear the temporary variables
```

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

The function first sets the minimum number to satisfy the bits. It then makes a random number between 0 to (2^(bits) -1) with the state (initialized in randstate) then adds the minimum number. After that the function checks whether the number is prime or not using the is_prime() function mentioned earlier. If yes, then finalize the number, if no, then make another random number between 0 to (2^(bits) -1) with the state (initialized in randstate) then add the minimum number until it's prime.

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

This function first initializes the copy of the given parameters so it's easier to change them without affecting the real variables. It then makes a while loop checking if the exponent is equal to 0 or not, if not, multiply the result by the exponent, then subtract 1 to the exponent, if yes, end the while loop and use modulo on the result with the given modulus. After that, clears all the temporary variables.

Pseudo Code:

```
Initialize variables for the computation
Set a while loop with condition if exponent > 0
Compute the result using repeated multiplication
Compute modulo of the result with the modulus
Clear all temporary variables
```

2. randstate.c

This section will take about the design of the file randstate.c

2.1 void randstate_init(uint64_t seed)

This function initializes the random state needed for SS key generation operations.

Pseudo Code:

Initialize the state

Set an initial seed value into state

Set the randomness with the seed

2.2 void randstate_clear(void)

This function frees any memory used by the initialized random state.

Pseudo Code:

Free the state

3. ss.c

This section will take about the design of the file ss.c

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

This function generates the components for a new SS key. It first gets the bits for prime 'p' and prime 'q' then compute them with make_prime() function, after that it computes n = p^2 * q.

Pseudo Code:

Initialize bits needed for prime 'p' and prime 'q'

Make prime 'p' and prime 'q'

Make 'n'

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

This function generates components for a new SS private key. It computes the private key 'd' by compute the inverse of n modulo $\lambda(pq) = \mathrm{lcm}(p-1,q-1)$

Pseudo Code:

Initialized needed variables

Get n = p^2 * q

Compute the inverse of n modulo $\lambda(pq) = \mathrm{lcm}(p-1,q-1)$

Clear all temporary variables

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

This function prints the public key to a file.

Pseudo Code:

Print public key to the pbfile with proper format

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

This function prints the private key to a file.

Pseudo Code:

Print private key to the pvfile with proper format

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

This function read the public key and transfer the information to the given parameters

Pseudo Code:

Read pbfile and set ‘n’ and ‘username’ to the proper value

3.6 void ss_read_priv(mpz_t pq, mpz_t d, FILE *pvfile)

This function read the private key and transfer the information to the given parameters

Pseudo Code:

Read pvfile and set ‘pq’ and ‘d’ to the proper value

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

This function encrypt a block of text using the public key

Pseudo Code:

Use exponential modulo to encrypt the text

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

This function encrypts information in ‘infile’ and prints the output to ‘outfile’. It first calculates block size k and allocates the memory needed for the block, then it sets the 0th byte to 0xFF. After that, it keeps looping and reading data at most (k-1) byte from the ‘infile’ and encrypts them using the function ss_encrypt() until the end of the file.

Pseudo Code:

```
Set block size k
Allocate memory for block k
Set the 0th byte to 0xFF
Make a while loop that check if it’s the end of file, if not, read the data in max (k-1) byte and encrypt them, if yes, end the loop
Clear all temporary variables and clear the memory for block k
```

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

This function decrypt a block of text using the private key

Pseudo Code:

```
Use exponential modulo to decrypt the text
```

3.10 void ss_decrypt_file(FILE *infile, FILE *outfile, const mpz_t d, const mpz_t pq)

This function decrypts information in ‘infile’ and prints the output to ‘outfile’. It first calculates block size k and allocates the memory needed for the block. After that, it keeps looping and reading data in block k byte from the ‘infile’ and decrypts them using the function ss_decrypt() until the end of the file.

Pseudo Code:

```
Set block size k
Allocate memory for block k
Set the 0th byte to 0xFF
Make a while loop that check if it’s the end of file, if not, read the data in max (k-1) byte and decrypt them, if yes, end the loop
Clear all temporary variables and memory for block k
```

4. keygen.c

This main file will generate the public key and private key and print to 2 files. It first sets up the getopt to get user input. After that it initializes the random state using randstate_init(), using the set seed. It then makes the public and private keys using ss_make_pub() and ss_make_priv(). The code then gets the current user’s name as a string. In the end, it writes the computed public and private key to their respective files.

Pseudo Code:

```
Initialized all necessary variables
Setup getopt to get user input
Initialize randstate and the seed
Get username
Call ss_make_pub() and ss_make_priv function to create the keys and output them to files
Print verbose if called
Clear all temporary variables
```

5. encrypt.c

This main file will encrypt data from the input file and put the encrypted data into an output file using the public key. It first checks whether the input, output, and public key files are available and accessible. It then calls the function ss_encrypt_file() to encrypt the data.

Pseudo Code:

```
Initialized all necessary variables
Setup getopt to get user input
Check the input, output, and public key files
Call function ss_encrypt_file()
Print verbose if called
Clear all temporary variables
```

6. decrypt.c

This main file will decrypt data from the input file and put the decrypted data into an output file using the private key. It first checks whether the input, output, and private key files are available and accessible. It then calls the function ss_decrypt_file() to decrypt the data.

Pseudo Code:

```
Initialized all necessary variables
Setup getopt to get user input
Check the input, output, and private key files
Call function ss_decrypt_file()
Print verbose if called
Clear all temporary variables
```