Kirk Perez
1918126
2/9/2023
Assignment 5

General Idea:
In this assignment, 3 main programs will be created: keygen, encrypt and decrypt. Two libraries, SS, and number theory will be implemented too. The libraries and modules are used to help reproduce the math that makes the 3 main programs. Keygen produces SS public & private keys. Encrypt encrypts files using a public key and the decrypt program is able to decrypt the said file using the corresponding private key that was produced.

Pseudocode:

keygen.c
Command Line Options

- 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: Display help message detailing program usage

Open the public and private key files using fopen(). If it fails, print an error message and exit.
Call fchmod() and fileno() to set private key file permissions to 0600 (enables read/write for only user)
Initialize seed to randstate_init()
Call ss_make_pub() and ss_make_priv() to make public and private keys
Call getenv() to get username as string
Write public key to public file and private key to private file
If verbose output is enabled print with \n:

1. Username
2. the first large prime p
3. the second large prime q
4. the public key n
5. the private exponent d
6. the private modulus pq

Close public key file
Call randstate_clear()
clear all mpz_t variables used

encrypt.c

Command Line Options

- i: Specifies the input file to encrypt (DEFAULT: stdin)
- o: Specifies the input file to encrypt (DEFAULT: stdout)
- n: Specifies the file containing the public key (DEFAULT: ss.pub)
- v: Enables verbose output
- h: Display help message detailing program usage

Open the public key file using fopen(). If it fails, print an error message and exit.
Read the public key from the public key file
If verbose output is enabled print with \n:

1. Username
2. the public key n

Encrypt file using ss_encrypt_file()
Close public files
clear all mpz_t variables used

decrypt.c

Command Line Options

- i: Specifies the input file to decrypt (DEFAULT: stdin)
- o: Specifies the input file to decrypt (DEFAULT: stdout)
- n: Specifies the file containing the private key (DEFAULT: ss.priv)
- v: Enables verbose output
- h: Display help message detailing program usage

Open the public key file using fopen(). If it fails, print an error message and exit.
Read the private key from the private key file
If verbose output is enabled print with \n:

1. the private modulus pq
2. the private key d

Encrypt file using ss_decrypt_file()
Close private key file
clear all mpz_t variables used

numtheory.c

void pow_mod(mpz_t o, const mpz_t a, const mpz_t d, const mpz_t n)
Initialize mpz_t v, p, and temp
Set v to 1 and p to base
While exponent > 0:
    if(exponent is odd):

```
                Make v = v*p % modulus
        P = p*p % modulus
        Exponent = exponent/2
Return v

bool is_prime(const mpz_t n, uint64_t iters)
If n is 0 or 1
        Return false
If n is 2 or 3
        Return true
If n is even
        Return false


Make var r = n-1
While r is odd
        Divide until even

for(i=1 and i < k):
        Var a = random number between (2 to n-2)
        Var y = power_mod(a,r,n)
        if (y != 1 and y != n-1):
                lnt j = 1
                        while ( j<= s-1 and y != n-1):
                                Y = power_mod(y,2,n)
                                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)
Generate a random integer with  mpz_urandomb
If the number is not prime
        Call make prime

void gcd(mpz_t g, const mpz_t a, const mpz_t b)
while (b != 0):
        G = b
        B = a%b
        A = b
```

void mod_inverse(mpz_t o, const mpz_t a, const mpz_t n)
Var r = n
Var r2 = a
Var t = 0
Var t2 = 1
While(r2 != 0)
    Var q = r/r2
    R = r2
    R2 = r - (q*r2)
    T = t2
    T2 = t - (q*t2)
if(r > 1)
    I = 1
    Return i
If(t < 0)
    T = t+n
Return t

randstate.c
void randstate_init(uint64_t seed):
Make a global variable named state using mt(seed)
Call gmp_randinit_mt()
Call gmp_randseed_ui()

void randstate_clear(void)
Call gmp_randclear()

ss.c
void ss_make_pub(mpz_t p, mpz_t q, mpz_t n, uint64_t nbits, uint64_t iters)
Initialize var p and q by using make_prime()
Set bits for pbits equal to a random number in the range [nbits/5,(2×nbits)/5)
Qbits = Make leftover bits
Call make_prime for p and q
Make var for p-1 and q-1
P = p1 % p
Q = q1 % q

While p1 or q1 are not 0
    Call make_prime for p and q
    Make var for p-1 and q-1
    P = p1 % p

Q = q1 % q
N = p*p*q

void ss_write_pub(const mpz_t n, const char username[], FILE *pbfile)
If in pbfile
Write n as a hex to pbfile
Write user to pbfile

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

void ss_make_priv(mpz_t d, mpz_t pq, const mpz_t p, const mpz_t q)
Pq = p*q
N = p*p*q
Make variables for p-1 and q-1
lcm = (p-1 * q-1)/ gcd(p-1, q-1)
d = 1/(n % lcm)

void ss_write_priv(const mpz_t pq, const mpz_t d, FILE *pvfile)
If in pvfile
Write pq to pvfile\n
Write d to pbfile\n

void ss_read_priv(mpz_t pq, mpz_t d, FILE *pvfile)
Read the private SS key from pbfile
If in pvfile
Read pq
Read d

void ss_encrypt(mpz_t c, const mpz_t m, const mpz_t n)
Encrypt m by E(m) = c= m^n (mod n)
Call pow_mod(c, m, n, n)

void ss_encrypt_file(FILE *infile, FILE *outfile, const mpz_t n)
Set block size, k, to be (log2(n)-1)/8
Malloc up to size k of type (uint8_t *)
Set the zeroth byte of the block to 0xFF

while(there are unprocessed bytes in the infile)
Initialize counter variable to 0 in order to see how many bytes read

for(count < k-1)
Place read bytes into allocated block, starting at index 1 and make the first block into mpz_t m
Convert read bytes by using mpz_import()
Encrypt m with ss_encrypt()
Write result to outfile as hexstring\n

void ss_decrypt(mpz_t m, const mpz_t c, const mpz_t d, const mpz_t pq)
Compute m
M = c^d % n
Call pow_mod(m, c, d, pq)

void ss_decrypt_file(FILE *infile, FILE *outfile, const mpz_t d, const mpz_t pq)
Set block size, k, to be (log2(n)-1)/8
Malloc up to size k of type (uint8_t *)
Set the zeroth byte of the block to 0xFF

while(there are unprocessed bytes in the infile)
Scan bytes and save as a hexstring as mpz_t c
Convert c into bytes using mpz_export()
Var j = number of bytes converted
Write j-1 bytes in starting from index 1 of array

Deliverables:

decrypt.c
– Contains the implementation and main() for decrypt

encrypt.c
– Contains the implementation and main() for decrypt

keygen.c
– Contains the implementation and main() for decrypt

numtheory.c
– Contains the implementation of the number theory functions

numtheory.h
– Interface for number theory functions

randstate.c
– Contains the implementation of random state for SS library and numtheory functions

randstate.h
– Interface for initializing and clearing the random state

ss.c
– Contains the implementation for the SS library

ss.h
– Interface for the SS library

readme.md
Makefile
DESIGN.pdf
WRITEUP.pdf