Design Document - Assignment 5

Hsiang Yun Lu

February 26, 2023

1 Purpose of Programs

The three main program will be the implementation of public key cryptography using Schmidt-Samoa (SS) algorithm. The `keygen.c` is a key generator that produces SS public and private key pairs. The `encrypt.c` program is an encryptor that encrypts files using a public key, while the `decrypt.c` program decrypts the encrypted files using the corresponding private key.

The `randstate.c` contains the implementation of a random state module. The `numtheory.c` program contains the implementations of the number theory functions behind SS. The `ss.c` contains implementations of routines for SS. The above libraries and module will be used in the three main programs.

2 Files To Be Included in Directory *asgn3*

- `decrypt.c`
  - This file contains the implementation and main() function for the `decrypt` program.
- `encrypt.c`
  - This file contains the implementation and main() function for the `encrypt` program.
- `keygen.c`
  - This file contains the implementation and main() function for the `keygen` program.
- `numtheory.c and numtheory.h`
  - The `numtheory.c` includes the implementations of the number theory functions.
  - The `numtheory.h` specifies the interface of the number theory functions.
- `randstate.c and randstate.h`

  - The `randstate.c` includes the implementations of the random state interface for the SS library and number theory functions.
  - The `randstate.h` specifies the interface for initializing and clearing the random state.

- `ss.c and ss.h`
  - The `ss.c` includes the implementations of the SS library.
  - The `ss.h` specifies the interface for the SS library.

- `Makefile`
  - This file compiles the programs and builds the `encrypt`, `decrypt`, and `keygen` executables.

- `README.md`
  - This file describes how to use all the programs and `Makefile`, including explanations on command-line options

- `DESIGN.pdf`
  - This file describes the design and design process for all the programs
  - Include pseudocode

- `WRITEUP.pdf`
  - Things learned in this assignment in detail
  - discussion on the applications of public-private cryptography and how it influences the world today

3 Design/Structure of `randstate.c`

void randstate_init(uint64_t seed)

    Initializes the global random `state`
    Set initial seed to `state`

void randstate_clear(void)

    Free all memory occupied by `state`

4 Design/Structure of numtheory.c

```
bool is_odd(mpz_t d)
    initialize mpz_t x, o
    x ← 1
    o ← d & x
    if o == 1
        Free the mpz_t variables
        return true
    else
        Free the mpz_t variables
        return false
```

```
bool is_even(mpz_t d)
    initialize mpz_t x, o
    x ← 1
    o ← d & x
    if o == 0
        Free the mpz_t variables
        return true
    else
        Free the mpz_t variables
        return false
```

```
void gcd(mpz_t g, mpz_t a, mpz_t b)
    initialize mpz_t t
    while b not 0
        t ← a
        a ← b
        b ← t % b
    g ← a
    Free the mpz_t variables
```

```
void mod_inverse(mpz_t o, mpz_t a, mpz_t n)
    initialize mpz_t r, rP, t, tP, q, tmp, qrP, qtP
    r ← n
    rP ← a
    tP ← 1
    while rP not 0
        q ← r / rP
        tmp ← rP
        qrP ← q × rP
        rP ← r - qrP
        r ← tmp
        tmp ← tP
        qtP ← q × tP
        tP ← t - qtP
        t ← tmp
    if r > 1
        return 0
    else
        if t < 0
            o ← t + n
        else
            o ← t
    Free the mpz_t variables

void pow_mod(mpz_t o, mpz_t a, mpz_t d, mpz_t n)
    initialize mpz_t v, p, vp, pp
    v ← 1
    p ← a
    while d > 0
        if is_odd(d)
            vp ← v × p
            v ← vp % n
        pp ← p × p
        p ← pp % n
    o ← v
    Free the mpz_t variables

void get_d_r(mpz_t d, mpz_t r, mpz_t n)
    d ← n
    r ← 0
    while is_even(d)
        d ← d / 2
        r ← r + 1

bool witness(mpz_t a, mpz_t n)
    initialize mpz_t d, r, x, y, k, l
    get_d_r(d, r, n - 1)
    pow_mod(x, a, d, n)
    for i = 0 to r-1
        k ← 2
        pow_mod(y, x, k, n)
        l ← n - 1
        if y == 1 and x not 1 and x not 0
            retuen true
        x ← y
        if x not 1
            Free the mpz_t variables
            retuen true
        else
            Free the mpz_t variables
            retuen false

bool is_prime(mpz_t n, uint64_t iters)
    initialize mpz_t md, a, nt
    md ← n % 2
    nt ← n - 1
    if n < 2 or (n != 2 and n % 2 == 0)
        return false
    if n == 2 or n == 3
        return true
    for i = 0 to iters - 1
        a ← randome mpz_t integer in range [0, nt]
        a ← a + 2
        if witness(a, n)
            return false
    Free the mpz_t variables
    return true

void make_prime(mpz_t p, uint64_t bits, uint64_t iters)
    initialize mpz_t r, g, bt, tp
    tp ← 1
    do
        g ← a random mpz_t integer
        bt ← tp « b
        bt ← bt - 1
        r ← g & bt
    while not is_prime(r, 50)
    p ← r
    Free the mpz_t variables
```

5 Design/Structure of `ss.c`

```
void ss_make_pub(mpz_t p, mpz_t q, mpz_t n, uint64_t nbits, uint64_t iters)
    low ← nbits / 5
    high ← (2 × nbits) / 5
    p_bit ← random() % (high - low) + low
    q_bit ← nbits - 2 × p_bit
    make_prime(p, p_bit, iters)
    make_prime(q, q_bit, iters)
    while p == q or (q - 1) % p == 0 or (p - 1) % q == 0
        make_prime(q, q_bit, iters)
    n ← p × p × q
```

```
void ss_make_priv(mpz_t d, mpz_t pq, mpz_t p, mpz_t q)
    initialize mpz_t p_, q_, g, ab, n
    p_ ← p - 1
    q_ ← q - 1
    ab ← p_ × q_
    gcd(g, p_, q_)
    pq ← ab / g
    n ← p × p × q
    mod_inverse(d, n, pq)
```

```
void ss_write_pub(mpz_t n, char username[], FILE *pbfile)
    gmp_fprintf(print n to pbfile)
    fprintf(print username to pbfile)
```

```
void ss_write_priv(mpz_t pq, mpz_t d, FILE *pvfile)
    gmp_fprintf(print pq to pbfile)
    gmp_fprintf(print d to pbfile)
```

```
void ss_read_pub(mpz_t n, char username[], FILE *pbfile)
    gmp_fscanf(read pbfile and assign to n)
    fscanf(read pbfile and assign to username)
```

```
void ss_read_priv(mpz_t pq, mpz_t d, FILE *pvfile)
    gmp_fscanf(read pvfile and assign to pq)
    gmp_fscanf(read pvfile and assign to d)
```

```
void ss_encrypt(mpz_t c, mpz_t m, mpz_t n)
    c ← $m^n$ % n
```

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

```
    initialize mpz_t k, m, c
    k ← ((log2(n) / 2) - 1) / 8
    *block ← calloc(k, sizeof(uint8_t))
    block[0] ← 0xFF
    if (Read at most k - 1 bytes in from infile into block starting block[1])
        mpz_import(convert the read bytes into m)
        ss_encrypt(c, m, n)
        gmp_fprintf(print c to outfile)

void ss_decrypt(mpz_t m, mpz_t c, mpz_t d, mpz_t pq)
    m ← $c^d$ % pq

void ss_decrypt_file(FILE *infile, FILE *outfile, mpz_t d, mpz_t pq)
    input ← 0
    initialize mpz_t k, m, c
    k ← (log2(pq) - 1) / 8
    *block ← calloc(k, sizeof(uint8_t))
    while true
        input ← fscanf(Read in a hexstring from infile and save as c)
        if input == 1
            ss_decrypt(m, c, d, pq)
            j ← mpz_sizeinbase(m, 2) / 8
            mpz_export(convert m into bytes and store them in the allocated block)
            fprintf(print out j - 1 bytes starting from block[1] to outfile)
        else if input == End of File
            break
```

6 Design/Structure of `keygen.c`

DEFINE command-line options “hvb:i:n:d:s:”

```
usage(program)
    print(program synopsis and usage)


main(arguments)
    opt ← 0
    bits ← 256
    iters ← 50
    s ← false
```

```
seed ← 0
verbose ← false
char *pb ← “ss.pub”
char *pv ← “ss.priv”
FILE *pbptr
FILE *pvptr

while opt ← here is at least an argument
    switch (opt)
        case argument “b”: bits ← number converted from read-in string
            break
        case argument “i”: iters ← number converted from read-in string
            break
        case argument “s”:
            s ← true
            seed ← number converted from read-in string
            break
        case argument “v”
            verbose ← true
            break
        case argument “n”
            pb ← optarg
            break
        case argument “d”
            pv ← optarg
            break
        case argument “h” do usage(argv[0])
            EXIT
        default do usage(argv[0])
            return EXIT_FAILURE

pbptr ← open pb file for write
pvptr ← open pv file for write
    if pbptr does not exist
        PRINT ERROR
        EXIT
    else if pvptr does not exist
        PRINT ERROR
        EXIT

fchmod(set the private key file permission to 0600)
```

```
if s is false
    randstate_init(initialize gmp random state using seed time(NULL))     else
    randstate_init(initialize gmp random state using user specified seed)
initialize mpz_t p, q, n, d, pq
ss_make_pub(p, q, n, bits, iters)
ss_make_priv(d, pq, p, q)

char *user ← get username

ss_write_pub(n, user, pbptr)
ss_write_priv(pq, d, pvptr)

if verbose is true
    print username
    print p to STDOUT
    print q to STDOUT
    print n to STDOUT
    print pq to STDOUT
    print d to STDOUT

close pbptr
close pvptr

clear mpz_t p, q, n, d, pq

RETURN 0
```

7 Design/Structure of `encrypt.c`

DEFINE command-line options “hvo:i:n:”

```
usage(program)
    print(program synopsis and usage)


main(arguments)
    opt ← 0
    verbose ← false
    inp ← false
    out ← false
```

```
char *pb ← “ss.pub”
char *input
char *output
FILE *finptr
FILE *foutptr
char user[]

initialize mpz_t n

while opt ← here is at least an argument
    switch (opt)
        case argument “n”:
            pb ← optarg
            break
        case argument “i”
            inp ← true
            input ← optarg
            break
        case argument “o”
            out ← true
            output ← optarg
            break
        case argument “v”
            verbose ← true
            break
        case argument “h” do usage(argv[0])
            EXIT
        default do usage(argv[0])
            return EXIT_FAILURE

pbptr ← open pb file
    if pbptr does not exist
        PRINT ERROR
        EXIT
    else
        ss_read_pub(Read in public key and username, store them in n and user, respectively)

finptr ← open input file to be encrypted
    if finptr does not exist
        PRINT ERROR
        EXIT
    foutptr ← open output file to write

    if verbose is true
        print username to STDOUT
        print n to STDOUT

    if inp is false and out is false
        ss_encrypt_file(read from STDIN and output to STDOUT)
    else if inp is false and out is true
        ss_encrypt_file(read from STDIN and output to FILE)
    else if inp is true and out is true
        ss_encrypt_file(read from FILE and output to FILE)
    else if inp is true and out is false
        ss_encrypt_file(read from FILE and output to STDOUT)

    close pbptr
    close finptr
    close foutptr

    clear mpz_t n

    RETURN 0
```

8 Design/Structure of `decrypt.c`

```
DEFINE command-line options “hvo:i:n:”

usage(program)
    print(program synopsis and usage)


main(arguments)
    opt ← 0
    verbose ← false
    inp ← false
    out ← false
    char *pb ← “ss.priv”
    char *input
    char *output
```

```
FILE *finptr
FILE *foutptr

initialize mpz_t d, pq

while opt ← here is at least an argument
    switch (opt)
        case argument “n”:
            pv ← optarg
            break
        case argument “i”
            inp ← true
            input ← optarg
            break
        case argument “o”
            out ← true
            output ← optarg
            break
        case argument “v”
            verbose ← true
            break
        case argument “h” do usage(argv[0])
            EXIT
        default do usage(argv[0])
            return EXIT_FAILURE

pvptr ← open pv file
    if pvptr does not exist
        PRINT ERROR
        EXIT
    else
        ss_read_priv(Read in public key and store them in pq and d)

finptr ← open input file to be decrypted
    if finptr does not exist
        PRINT ERROR
        EXIT

foutptr ← open output file to write

if verbose is true
    print username to STDOUT
    print n to STDOUT

if inp is false and out is false
    ss_decrypt_file(read from STDIN and output to STDOUT)
else if inp is false and out is true
    ss_decrypt_file(read from STDIN and output to FILE)
else if inp is true and out is true
    ss_decrypt_file(read from FILE and output to FILE)
else if inp is true and out is false
    ss_decrypt_file(read from FILE and output to STDOUT)

close pbptr
close finptr
close foutptr

clear mpz_t d, pq

RETURN 0
```

9 Credit

- Dev's section on Feb. 14th.
- Some functions were modified from the *primes.py* in resources
- Some functions were modified from the *test-prime-ss.c* in resources
- Some functions were modified from the *ss.py* in resources