DESIGN

Wenyu Liang

February 27, 2023

1 Program Description

There are three programs for this assignment. They are keygen, encrypt and decrypt. The keygen program will be in charge of key generation, producing SS public and private key pairs. The encrypt program will encrypt files using a public key, and the decrypt program will decrypt the encrypted files using the corresponding private key.

2 Files to be included in directory asgn5

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.

3 Structure

1. **numtheory.c:** This file will implement all the number theory functions
2. **randstate.c:** Initialize and clear the random state
3. **keygen.c:** Implement public key and private key generation
4. **ss.c:** Implement encryption and decryption

4 Pseudocode

1. **randstate.c**

```
//following functions are learned from
//the given python implementation
void gcd(g, a, b) {
    while (b != 0) {
        t=b
        b=(a%b)
        a=t
    }
    g=a
}

void mod_inverse(o, a, n) {
    r=n
    rP=a
    t=0
    tP=1
    while (rP != 0) {
        q=r//rP
        rT=r
        r=rP
        rT2=q/rP
        rP=rT-rT2
        tT=t
        t=tP
        tT2=q*tP
        tP=tT*tT2
    }
    if r>1:
        o=0
    if t<0:
        t = t+n
}
```

```
void pow_mod(o, a, d, n) {
    o=1
    p=a
    while (d>0) {
        if (is_odd(d)) {
            v=(v*p)%n
        }
        p=(p*p)%n
        d=d/2
    }
}

void get_d_r(n, d, r) {
    /* Factors n into the form d * 2 ** r. */
    d=n
    r=0
    while (is_even(d)) {
        d //= 2
        r=r+1
    }
}

bool witness(mpz_t a, mpz_t n) {
    n_minus_1=n-1
    get_d_r(n_minus_1, d, r); // Factor n into d * 2**r + 1
    pow_mod(x, a, d, n);
    two=2
    for (i in range(0,r) {
        pow_mod(y, x, two, n);
        if (y==1 && x!=1 && x!=n_minus_1) {
            return true;
        }
        x=y
    }
    return x!= 1;
}

bool is_prime(mpz_t n, uint64_t iters) {
    if (n < 2 || (n!=0 && is_even(n))) {
        return false;
    }
    if (n==2 || n==3) {
        return true;
    }

    n_minus_3=n-3
```

```
    for (i in range(iters)) {
        mpz_urandomm(a, state, n_minus_3);
        a=a+2/ Euler witness (or liar)
        if (witness(a, n)) {
            return false;
        }
    }
    return true;
}

void make_prime(mpz_t p, uint64_t bits, uint64_t iters) {
    while (1) {
        mpz_rrandomb(p, state, bits);
        if (is_prime(p, iters)) {
            break;
        }
    }
}
```

2. **ss.c**

```
static void lcm(mpz_t p, mpz_t q, mpz_t o) {
    o = (p * q)// gcd(a, b)
}

void ss_make_pub(mpz_t p, mpz_t q, mpz_t n, uint64_t nbits, uint64_t iters)
    lower_p = nbits / 5;
    upper_p = nbits * 2 / 5;
    //making sure the bits_p is in the given range
    uint64_t bits_p = random() % (upper_p - lower_p) + lower_p;
    make_prime(p, bits_p, iters);
    p_sqr = p * p
    bits_q = nbits - 2 * bits_p;
    make_prime(q, bits_q, iters);
    n = p_sqr * q
}

void ss_make_priv(mpz_t d, mpz_t pq, mpz_t p, mpz_t q) {
    // Calculate lambda(n) = lcm(p-1, q-1)
    p_minus_one = p - 1
    q_minus_one = q - 1
    lcm(p_minus_one, q_minus_one, lambda);
    pq =  p * q
    n =  pq * p
    // Calculate d = inverse of n modulo lambda(n)
    mod_inverse(d, n, lambda);
```

```
}

void ss_write_pub(mpz_t n, char username[], FILE *pbfile) {
    gmp_fprintf(pbfile, "%Zx\n%s\n", n, username);
}

void ss_read_pub(mpz_t n, char username[], FILE *pbfile) {
    char n_str[1024], username_str[_POSIX_LOGIN_NAME_MAX];

    // Read in n and username as strings
    if (fgets(n_str, sizeof(n_str), pbfile) == NULL) {
        fprintf(stderr, "Error reading SS public key\n");
        exit(1);
    }
    if (fgets(username_str, sizeof(username_str), pbfile) == NULL) {
        fprintf(stderr, "Error reading username\n");
        exit(1);
    }

    // Parse n as a hexadecimal number
    if (gmp_sscanf(n_str, "%Zx", n) != 1) {
        fprintf(stderr, "unable to parse public key\n");
        exit(1);
    }

    // Copy username string to username argument
    strncpy(username, username_str, sizeof(username_str));
    username[sizeof(username_str) - 1] = '\0';
    // Remove trailing newline from username
    if (strlen(username) > 0 && username[strlen(username) - 1] == '\n') {
        username[strlen(username) - 1] = '\0';
    }
}

void ss_write_priv(mpz_t pq, mpz_t d, FILE *pvfile) {
    gmp_fprintf(pvfile, "%Zx\n%Zx\n", pq, d);
}

void ss_read_priv(mpz_t pq, mpz_t d, FILE *pvfile) {
    char pq_str[1024], d_str[1024];

    if (fgets(pq_str, sizeof(pq_str), pvfile) == NULL) {
        fprintf(stderr, "Error reading private modulus\n");
        exit(1);
    }
    if (fgets(d_str, sizeof(d_str), pvfile) == NULL) {
```

```
        fprintf(stderr, "Error reading private exponent\n");
        exit(1);
    }

    if (gmp_sscanf(pq_str, "%Zx", pq) != 1) {
        fprintf(stderr, "unable to parse private modulus\n");
        exit(1);
    }
    if (gmp_sscanf(d_str, "%Zx", d) != 1) {
        fprintf(stderr, "unable to parse private exponent\n");
        exit(1);
    }
}

void ss_encrypt(mpz_t c, mpz_t m, mpz_t n) {
    pow_mod(c, m, n, n);
}

//
// Encrypt an arbitrary file
//
// Provides:
//  fills outfile with the encrypted contents of infile
//
// Requires:
//  infile: open and readable file stream
//  outfile: open and writable file stream
//  n: public exponent and modulus
//
void ss_encrypt_file(FILE *infile, FILE *outfile, mpz_t n) {
    uint64_t k = ((mpz_sizeinbase(n, 2) - 1) * 0.5 - 1) / 8;
    //uint8_t is 1 byte
    uint8_t *block = (uint8_t *) malloc(k * sizeof(uint8_t));
    block[0] = 0xFF;
    while (!feof(infile)) {
        // Read at most k-1 bytes into block
        size_t j = fread(&block[1], sizeof(uint8_t), k - 1, infile);

        // Convert block to mpz_t m
        mpz_t m;
        mpz_init(m);
        mpz_import(m, j + 1, 1, sizeof(uint8_t), 1, 0, block);

        // Encrypt m with ss_encrypt()
        mpz_t c;
        mpz_init(c);
```

```
        ss_encrypt(c, m, n);
        // Write encrypted number to outfile
        gmp_fprintf(outfile, "%Zx\n", c);

        // Clear m and c
        mpz_clear(m);
        mpz_clear(c);
    }

    // Free block array
    free(block);
}

void ss_decrypt(mpz_t m, mpz_t c, mpz_t d, mpz_t pq) {
    pow_mod(m, c, d, pq);
}

// Decrypt a file back into its original form.
//
// Provides:
//  fills outfile with the unencrypted data from infile
//
// Requires:
//  infile: open and readable file stream to encrypted data
//  outfile: open and writable file stream
//  d: private exponent
//  pq: private modulus
//
void ss_decrypt_file(FILE *infile, FILE *outfile, mpz_t d, mpz_t pq) {
    uint64_t k = (mpz_sizeinbase(pq, 2) - 2) / 8;
    uint8_t *block;
    block = (uint8_t *) malloc(k * sizeof(uint8_t));
    // Loop through each line in infile
    char hexstr[k * 10];
    while (fscanf(infile, "%s\n", hexstr) != EOF) {
        // Convert hexstring to mpz_t
        mpz_t c;
        mpz_init(c);
        mpz_set_str(c, hexstr, 16);
        // Decrypt c to get m
        mpz_t m;
        mpz_init(m);
        ss_decrypt(m, c, d, pq);
        // Export m to block
        size_t j;
        mpz_export(&block[0], &j, 1, 1, 0, 0, m);
```

```
            // Starting from 1 to remove 0xFF
            fwrite(&block[1], 1, j - 1, outfile);

            // Clean up
            mpz_clear(c);
            mpz_clear(m);
        }

        // Free block
        free(block);
    }
```