C语言实现RSA非对称对称算法

学号：18342069

姓名：罗炜乐

使用方法

```
cd hw2_RSA && make
cd bin
./rsa
```

文件结构

```
hw1_DES
    |
    --- README.md
    |
    --- include
    |   |
    |   --- rsa.h
    |
    --- src
    |   |
    |   --- rsa.c
    |   |
    |   --- main.c
    |
    --- bin
        |
        --- rsa
```

解码原理描述

欧拉定理

$$\text{对互素的 } a \text{ 和 } m\text{, 有 } a^{\phi(m)} \equiv 1(\bmod m)$$

推论

$$\text{给定满足 } N = pq \text{ 的两个不同素数 } p \text{ 和 } q \text{ 以及满足 } 0 < n < N \text{ 的 整数 } n, k \text{ 是正整数，有 } n^{k\varphi(N)+1} \equiv n(\bmod N).$$

因此

$$\text{由于 } ed \equiv 1(\bmod \varphi(N))\text{, 即 } ed = k\varphi(N) + 1\text{, 故}$$

$$n^{ed} = n^{k\varphi(N)+1} \equiv n(\bmod N), \text{ 即 } n^{ed} \bmod N = n \bmod N.$$

现在我们有

$$c = n^e \bmod N, n' = c^d \bmod N$$

应用模算术运算规则得到

$$n' = c^d \bmod N = (n^e)^d \bmod N = n \bmod N$$

数据结构

本次实验的实现依赖于GMP大数库，所有的大数运算都使用该库。该库使用动态分配内存的方法存储大数及其运算。

生成密钥

RSA 算法中 p 和 q 的选择流程

1. 确定 RSA 所要求 N 的位数 k。 $k = 1024、2048、3072、4096\ldots$
2. 随机选择一个位数为 $(k+1)/2$ 的素数 p。即 $p \in \left[2^{(k+1)/2-1}, 2^{(k+1)/2} - 1\right]$
3. 选择一个位数为 $k-(k+1)/2 = (k-1)/2$ 的素数 q。
4. 求 $|p-q|$; 如果 $\log|p-q|$ 过小，则返回 (2)， 重新选取 p_0
5. 计算 $N = pq$, 确认 N 的位数为 $k\left(N \in \left[2^{k-1}, 2^k - 1\right]\right)$； 否则返回 (2) 重新选取 p。
6. 计算 N 的 NAF 权重 ; 如果权重过小 , 则返回 (2), 重新选取 p 。
7. 计算 $\varphi(N)$, 选择公钥指数 $e, 2^{16} < e < \varphi(N)$ 且 $\gcd(e, \varphi(N)) = 1$。PKCS#1 建议选择素数 $e = 2^{16} + 1 = 65537$ 。
8. 求 e 的模 $\varphi(N)$ 逆元 d 作为私钥指数 ; 如果 d 过小，则返回 (2), 重新选取 p 。
9. p 和 q 选择成功，销毁 p、q, 返回参数 N、e、d。

代码如下所示

```
/**
* @brief 生成一对公私钥
* @param k 钥匙的位数
* @param n 密钥中 n 的引用
* @param e 公钥中 e 的引用
* @param d 私钥中 e 的引用
*/
void generate_key(int k, mpz_t n, mpz_t e, mpz_t d) {
    gmp_randstate_t grt;
    gmp_randinit_default(grt);
    gmp_randseed_ui(grt, time(NULL));
    mpz_t p_lb, q_lb, n_lb, p_hb, q_hb, p, q, f, temp;
    mpz_init(p_lb);
    mpz_init(q_lb);
    mpz_init(p_hb);
    mpz_init(q_hb);
    mpz_init(n_lb);
    mpz_init(p);
    mpz_init(q);
    mpz_init(f);
    mpz_init(temp);
    mpz_ui_pow_ui(p_lb, 2, (k + 1) / 2 - 1);
    mpz_ui_pow_ui(q_lb, 2, k - (k + 1) / 2 - 1);
    mpz_ui_pow_ui(n_lb, 2, k - 1);
    mpz_ui_pow_ui(p_hb, 2, (k + 1) / 2);
    mpz_ui_pow_ui(q_hb, 2, k - (k + 1) / 2);
    // step 1 是参数
    // strp 7
    mpz_set_ui(e, 65537);
    // step 3
    while (1) {
        mpz_urandomb(q, grt, (k - 1) / 2);
        if (mpz_cmp(q_lb, q) > 0) {
            mpz_add(q, q, q_lb);
        }
        mpz_nextprime(q, q);
        if (mpz_cmp(q, q_hb) >= 0) {
            continue;
        }
        break;
    }
```

```
    while (1) {
        // step 2
        mpz_urandomb(p, grt, (k + 1) / 2);
        if (mpz_cmp(p_lb, p) > 0) {
            mpz_add(p, p, p_lb);
        }
        mpz_nextprime(p, p);
        if (mpz_cmp(p, p_hb) >= 0) {
            continue;
        }

        // step 4
        if (mpz_cmp(p, q) >= 0) {
            mpz_sub(temp, p, q);
        }
        else {
            mpz_sub(temp, q, p);
        }
        if (mpz_sizeinbase(temp, 2) <= ((k/2 - 100 < k / 3) ? k/2 - 100 : k / 3)) {
            continue;
        }

        // step 5
        mpz_mul(n, p, q);
        if (mpz_cmp(n, n_lb) < 0) {
            continue;
        }

        // step 8
        mpz_sub_ui(p, p, 1);
        mpz_sub_ui(q, q, 1);
        mpz_mul(f, p, q);
        mpz_invert(d, e, f);
        if (mpz_sizeinbase(d, 2) <= k / 2) {
            continue;
        }
        break;
    }
    mpz_clear(p_lb);
    mpz_clear(q_lb);
    mpz_clear(p_hb);
    mpz_clear(q_hb);
    mpz_clear(n_lb);
    mpz_clear(p);
    mpz_clear(q);
    mpz_clear(f);
    mpz_clear(temp);
}
```

加密和解密

核心

Bob 引用公钥 (e, N), 利用下面的公式, 将 n 加密为 c:

$$c = n^e \bmod N$$

Bob 算出 c 后可以将它经公开媒体传递给 Alice。

Alice 得到 Bob 的消息 c 后利用她的私钥 (d, N) 解码。 Alice 利用以下的公式将 c 转换为 n':

$$n' = c^d \bmod N$$

Alice 得到的 n' 就是 Bob 的 n, 因此可以将原来的信息 M 准确复原。

上述过程是对称的，可以公用同一个函数

```
/**
* @brief RSA 中对大数加解密算法
* @param key_n 公私钥的 n
* @param e 公钥中 e 或私钥中 d 的引用
* @param message 需要加解密的大数
```

```
 * @param result 加解密后的结果
 */
void rsa_adp_aep(mpz_t key_n, mpz_t e, mpz_t message, mpz_t result) {
    if (mpz_cmp_ui(message, 0) < 0 || mpz_cmp(message, key_n) >= 0) {
        fprintf(stderr, "message representative out of range\n");
        exit(1);
    }

    mpz_powm(result, message, e, key_n);
}
```

加密

1. 进行编码，填充字符串至 key_len 长度。构建 EM = 0x00 || 0x22 || PS || 0x00 || message
2. 将字符串转换为大数
3. 执行 RSA 核心，将大数加密
4. 将加密后的大数转换为字符串
5. 将原来的缓冲区换成新的缓冲区

```
/**
* @brief 对字符串 M(message) 加密
* @param key_n 公钥的 n
* @param public_key_e 公钥的 e
* @param key_len 密钥长度
* @param M 字符串缓冲区的地址，注意，如果缓冲区为 buffer[]，那应该传入&buffer，而且缓冲区必须用 malloc 分配，加密后该函数会释放原来的缓冲区并新分配一个缓冲区到M
*/
void rsa_encrypt(mpz_t key_n, mpz_t public_key_e, int key_len, char ** M) {
    int m_len = strlen(*M);
    if (m_len > key_len - 11) {
        fprintf(stderr, "message too long\n");
        exit(1);
    }

    // 进行编码，填充字符串至 key_len 长度
    char * EM = malloc(key_len + 1);
    EM[0] = 0x00;
    EM[1] = 0x02;
    srand((unsigned)time(NULL));
    for (int i = 0; i < key_len - m_len - 3; ++i) {
        EM[2 + i] = rand() % 255 + 1;
    }
    EM[key_len - m_len - 1] = 0x00;
    strcpy(EM + key_len - m_len, *M);

    mpz_t m;
    mpz_t c;
    mpz_init(m);
    mpz_init(c);

    // 将字符串转换为大数
    os2ip(EM, key_len, m);
    // 执行 RSA 核心，将大数加密
    rsa_adp_aep(key_n, public_key_e, m, c);
    // 将加密后的大数转换为字符串
    i2osp(c, key_len, EM);
    // 将原来的缓冲区换成新的缓冲区
    free(*M);
    *M = EM;
    mpz_clear(m);
    mpz_clear(c);
}
```

解密

1. 将字符串转换为大数
2. 执行 RSA 核心，将大数加密
3. 将加密后的大数转换为字符串
4. 检查解密的格式是否符合标准

5. 将原来的缓冲区换成新的缓冲区

```
/**
* @brief 对字符串 C(cipertext) 解密
* @param key_n 公钥的 n
* @param public_key_d 私钥的 d
* @param key_len 密钥长度
* @param C 字符串缓冲区的地址，注意，如果缓冲区为 buffer[]，那应该传入&buffer，而且缓冲区必须用 malloc 分配，加密后该函数会释放原来的缓冲区并新分配一个缓冲区到C
*/
void rsa_decrype(mpz_t key_n, mpz_t private_key_d, int key_len, char ** C) {
    if (key_len < 11) {
        fprintf(stderr, "decryption error\n");
        exit(1);
    }

    mpz_t m;
    mpz_t c;
    mpz_init(m);
    mpz_init(c);
    // 将字符串转换为大数
    os2ip(*C, key_len, c);
    // 执行 RSA 核心，将大数加密
    rsa_adp_aep(key_n, private_key_d, c, m);
    // 将加密后的大数转换为字符串
    i2osp(m, key_len, *C);
    mpz_clear(m);
    mpz_clear(c);
    // 检查解密的格式是否符合标准
    if ((*C)[0] != 0x00 || (*C)[1] != 0x02) {
        fprintf(stderr, "decryption error\n");
        exit(1);
    }

    int i = 2, ps_len = 0;
    for (; (*C)[i] != 0x00 && i < key_len; ++i, ++ps_len);
    ++i;
    if (ps_len < 8 || i >= key_len) {
        fprintf(stderr, "decryption error\n");
        exit(1);
    }

    char * message = malloc(key_len - i + 1);
    message[key_len - i] = 0;
    strncpy(message, *C + i, key_len - i);
    // 将原来的缓冲区换成新的缓冲区
    free(*C);
    *C = message;
}
```

编码

OS2IP

对于长度为 k 的 EM 存在如下格式 $X_0X_1\cdots X_{k-1}$　　得到明文大数 $M = X_0 * 256^{k-1} + X_1 * 256^{k-2} + X_2 * 256^{k-3} + \cdots + X_{k-2} * 256 + X_{k-1}$

```
/**
* @brief 将字符串转成一个大数
* @param str 要转换的字符串
* @param len 密钥长度
* @param result 转化后的大数结果的引用
*/
void os2ip(char * str, int len, mpz_t result) {
    mpz_t pow;
    mpz_init(pow);

    for (int i = 0; i < len; ++i) {
        mpz_set_ui(pow, 0);
        mpz_ui_pow_ui(pow, 256, len - i - 1);
        mpz_mul_ui(pow, pow, (uint8_t)str[i]);
        mpz_add(result, result, pow);
```

```
    }
    mpz_clear(pow);
}
```

I2OSP

$$C = X_0 * 256^{k-1} + X_1 * 256^{k-2} + X_2 * 256^{k-3} + \cdots + X_{k-2} * 256 + X_{k-1}$$ 逆向得到长度为 k 的密文 $X_0 X_1 \cdots X_{k-1}$

```
/**
* @brief 将大数转成一个字符串
* @param x 要转换的大数
* @param len 密钥长度
* @param result 转化后的字符串
*/
void i2osp(mpz_t x, int len, char * result) {
    mpz_t temp;
    mpz_init_set_ui(temp, 256);
    mpz_pow_ui(temp, temp, len);
    if (!(mpz_cmp(x, temp) < 0)) {
        fprintf(stderr, "integer too large\n");
        exit(1);
    }
    result[len] = 0;
    for (int i = len - 1; i >= 0; --i) {
        result[i] = 0;
        result[i] |= mpz_fdiv_q_ui(x, x, 256);
    }
}
```

编译结果

在本目录直接 make， 就可以得到可执行文件 ./bin/rsa

```
[luowle@VM_0_4_centos ~]$ cd homework/InformationSecurity/hw2_RSA/
[luowle@VM_0_4_centos hw2_RSA]$ make
gcc -std=c99 -g -lgmp -Iinclude src/main.c src/rsa.c -o bin/rsa -lm
[luowle@VM_0_4_centos hw2_RSA]$ ./bin/rsa
```

测试

使用如下函数进行测试

```
#include <stdio.h>
#include <string.h>
#include <malloc.h>
#include "rsa.h"

int main() {
    mpz_t n, e, d;
    mpz_init(n);
    mpz_init(e);
    mpz_init(d);
    generate_key(1024, n, e, d);
    gmp_printf("密钥 n: %ZX\n", n);
    gmp_printf("公钥 e: %ZX\n", e);
    gmp_printf("私钥 d: %ZX\n", d);
    char * message = malloc(100);
    strcpy(message, "Hello, world!\n");
    printf("加密前明文: %s", message);
    rsa_encrypt(n, e, 1024/8, &message);
    printf("加密后密文: %s\n", message);
    rsa_decrype(n, d, 1024/8, &message);
    printf("解密后明文: %s", message);

    free(message);
    mpz_clear(n);
    mpz_clear(e);
```

```
    mpz_clear(d);
}
```

测试结果如下

```
[luowle@VM_0_4_centos hw2_RSA]$ ./bin/rsa
密钥 n:
B70660683966D55B851DDD6E9EBD2A7B5D4D1D82C43748A95D0B48FE732FAB8FED5F31A17EA7B2D55D68360E81574EBAC7A5D48AC2F16E4A6DBE7
FA5A386288CACFC470EC90DAB76BD5B31611749C776DE0904987328250979DEEAB47C8A4E958C990ECB2B98E0C1346DE08F17D87E03EE8476E095
B06C7390FA068F9591BCFF
公钥 e: 10001
私钥 d:
AF1FD6D7633556293FF21792652933A0DEE18EBF34E8810A2D114342D50C63E84F84AA84901FFE29D23A889ED07BADC5628AF617DFF80B4404462
6F562C657BCC23CB059E02962749D6C52F8A88C08E95EF627EC66D645199FB96C19087FDDA77CAE114EE1238E3B2149B57479A38D277EB07B6232
BABC72CE3673F4A82E3481
加密前明文: Hello, world!
加密后密文: c
             Md脊+'X@GY6z>qY41*R.IY)
+EIUL6@!svL/i+%Ti{،1 -Б
解密后明文: Hello, world!
```

```
[luowle@VM_0_4_centos hw2_RSA]$ ./bin/rsa
密钥 n:
929F09D5E5F1B6171432BC05A597256121A34AE3A84EF89D04ABA9C8689D83D971226997FD0F3AF63437EE5E1E4114A74F30B0C5F8ACAE3A08140
1800910FD9A3FC7A32A1EC9A2E3F0B8FF11194AD17559E880BAD662DDAF640EACF1FB061A7F4B49A85637BFEA4D1795C15ADB651A29390FBBF0C7
71905C0E7A46C713EEE173
公钥 e: 10001
私钥 d:
14C2F061996DE09AE4014B1E81942576DB8C83CE38C6F5ED45FC08CE9801B5E9D45B4B187A7CED9C6F487C3857905F9C515A53ADB4C8BEE2F8C1D
C7484D42314BD369DCEF5F6B9CF59875B22CC8769E1DF456E62ECAF2512C34727A4CD5BFD1449AC558EC98D6B5E114C42F54483078867D5748B57
49D52F6B253913D57A6C01
加密前明文: Hello, world!
-nfA.&9o Q: ?(.@!V49(HX
          EZ0B2^IRzY;7@%ㅂ_v0=%70<~g2Hyu$*|#T'
解密后明文: Hello, world!
```

可见可以成功地加密解密，可以证明我的实现基本正确。