RSAES-PKCS1-v1_5 体系实现

原理说明

RSA是最广为使用的"非对称加密算法"。

密钥生成

1. **随机选择两个不相等的质数p和q。**
2. **计算p和q的乘积n。**
3. **计算n的欧拉函数φ(n)。**
4. **选择一个整数e，一般为65537，且e与φ(n) 互质。**
5. **计算e对于φ(n)的模反元素d。**
6. **将n和e封装成公钥，n和d封装成私钥。**

可靠性

大整数n的因数分解，是一件非常困难的事情，所以不能快速破解出私钥。

加密

待加密信息为m，密文为c，公钥为(n,e)

计算

$$m^e \equiv c \pmod{n}$$

解密

私钥为(n, d)

计算

$$c^d \equiv m \pmod{n}$$

解密证明

根据加密规则

$$m^e \equiv c \pmod{n}$$

得

$$c = m^e - kn$$

将c代入要要证明的那个解密规则：

$$(m^e - kn)d \equiv m \pmod{n}$$

化为

$$m^{h\varphi(n)+1} \equiv m \pmod{n}$$

（1）m与n互质。

根据欧拉定理，此时

$$m^{\varphi(n)} \equiv 1 \pmod{n}$$

得到

$$(m^{\varphi(n)})^h \times m \equiv m \pmod{n}$$

原式得到证明。

（2）m与n不是互质关系。

此时，由于n等于质数p和q的乘积，所以m必然等于kp或kq。

以 m = kp为例，考虑到这时k与q必然互质，则根据欧拉定理，下面的式子成立：

$$(kp)^{q-1} \equiv 1 \pmod{q}$$

写为

$$(kp)^{ed} = tq + kp$$

这时t必然能被p整除，即 t=t'p

$$(kp)^{ed} = t'pq + kp$$

因为 m=kp，n=pq，所以

$$m^{ed} \equiv m \pmod{n}$$

原式得到证明。

数据结构设计

rsa.h

```
struct triple {
    char x[KEY_LENGTH + 10];
    char y[KEY_LENGTH + 10];
    char d[KEY_LENGTH + 10];
};
```

triple结构体用来存储扩展欧拉定理计算过程中的数据存储(extEuclid函数)，其中x、y、d的含义与课件中的相同，使用char数组是为了存储大整数；

```
struct keyPair {
    char *N;
    char *d;
    int e;
};
```

keyPair结构体用来存储公私钥，(N, e)为公钥，(N, d)为私钥，采用char数组同样是为了存储大整数，e采用int是因为程序指定e固定为65537，不需要大整数的方式存储。

大整数存储(gmp.h)

大整数存储采用第三方库 `gmp.h` 实现，一般存储在 `mpz_t` 类型中，部分传参或存储的地方会将该类型转化为 `char` 数组，做大整数运算时则采用 `mpz_t` 类型，用 `gmp.h` 中的函数完成操作。

总体结构

- main.c： 调用相关函数，处理输入输出等；
- rsa.h： 相关函数的声明，相关结构体、常量的定义；
- rsa.c ： 相关函数的具体实现
- main： Linux下可运行的文件
- rsa.o/main.o： 中间文件

密钥生成

生成密钥对的rsaGenKey函数，主要进行生成质数、设置e、计算n、计算欧拉函数、计算模反元素；

> 注意：计算模反时可能需要加phiN以确保d大于0；

```
struct keyPair* rsaGenKey() {
    mpz_t* PandQ = rsaGenPrime();
    ...
    // 设置e为65537
    mpz_init_set_ui(e, 65537);

    // 计算n=p*q
    mpz_mul(N, PandQ[0], PandQ[1]);

    // 计算欧拉函数ϕ(N)=(p-1)*(q-1)
    mpz_sub_ui(PandQ[0], PandQ[0], 1);
    mpz_sub_ui(PandQ[1], PandQ[1], 1);
    mpz_mul(phiN, PandQ[0], PandQ[1]);

    // 计算数论倒数
    ...
    struct triple ee = extEuclid(cphiN, ce);
    mpz_t ze;
    mpz_init(ze);
    for (; mpz_cmp(d, ze) < 0; mpz_add(d, d, phiN)) {
    }
    ...
}
```

rsaGenKey函数中通过rsaGenPrime函数来生成质数，其中调用gmp.h的库函数来完成质数生成；

```
mpz_t* rsaGenPrime() {
    ...
    // 随机生成大整数
    mpz_urandomb(p, grt, KEY_LENGTH / 2);
    mpz_urandomb(q, grt, KEY_LENGTH / 2);
    ...
    // 使用GMP自带的素数生成函数
    mpz_nextprime(ref[0], p);
    mpz_nextprime(ref[1], q);
    ...
}
```

rsaGenKey函数中通过extEuclid函数来生成模反元素，extEuclid函数执行**扩展欧拉定理**，该函数是通过递归实现的；

```
struct triple extEuclid(char* a, char* b) {
    char x[KEY_LENGTH + 10];
```

```
    char y[KEY_LENGTH + 10];
    char d[KEY_LENGTH + 10];
    struct triple ee;
    ee.x[0] = '1';
    ee.x[1] = '\0';
    ee.y[0] = '0';
    ee.y[1] = '\0';
    strcpy(ee.d, a);
    if (strcmp(b, "0") == 0) {
        return ee;
    }

    mpz_t amb, ma, mb;
    mpz_init(amb);
    mpz_init_set_str(ma, a, BASE);
    mpz_init_set_str(mb, b, BASE);
    mpz_mod(amb, ma, mb);
    char mid[KEY_LENGTH + 10];
    mpz_get_str(mid, BASE, amb);
    // 递归
    ee = extEuclid(b, mid);
    strcpy(x, ee.y);

    mpz_div(ma, ma, mb);
    mpz_t ex, ey, my;
    mpz_init(my);
    mpz_init_set_str(ex, ee.x, BASE);
    mpz_init_set_str(ey, ee.y, BASE);
    mpz_mul(ma, ma, ey);

    mpz_sub(my, ex, ma);
    char temp[KEY_LENGTH + 10];
    mpz_get_str(temp, BASE, my);

    struct triple ref;
    strcpy(ref.x, x);
    strcpy(ref.y, temp);
    strcpy(ref.d, ee.d);

    mpz_clear(amb);
    mpz_clear(ma);
    mpz_clear(mb);
    mpz_clear(ex);
    mpz_clear(ey);
    mpz_clear(my);
    return ref;
}
```

加密

加密函数中的核心是`mpz_pown_ui`库函数，它完成幂运算和模运算；

```
char* rsaEncrypt(const char* plaintext, const char* key_n, int key_e) {
    mpz_t M, C, N;
    mpz_init_set_str(M, plaintext, BASE);
    mpz_init_set_str(N, key_n, BASE);
    mpz_init_set_ui(C, 0);

    mpz_powm_ui(C, M, key_e, N);  //使用GMP中模幂计算函数

    char* ref = (char*)malloc(sizeof(char) * (KEY_LENGTH + 10));
    mpz_get_str(ref, BASE, C);

    return ref;
}
```

解密

解密与加密过程类似，主要区别是采用公钥还是私钥，采用`mpz_powm`来计算模幂；

```
char* rsaDecrypt(const char* ciphertext, const char* key_n, const char* key_d) {
    mpz_t M, C, N, d;
    mpz_init_set_str(C, ciphertext, BASE);
    mpz_init_set_str(N, key_n, BASE);
    mpz_init_set_str(d, key_d, BASE);
    mpz_init(M);

    // 模幂计算
    mpz_powm(M, C, d, N);

    char* ref = (char*)malloc(sizeof(char) * (KEY_LENGTH + 10));
    mpz_get_str(ref, BASE, M);

    return ref;
}
```

编解码

编码

编码的核心是将字符转化为十进制ASCII码（三位），将转化后的数字依序顺序连接，形成一个大整数；

> 注意：由于gmp.h的需要，大整数存放在char数组中；

增加辅助函数`linkInt`来往串中添加特定的数字；

```
void linkInt(char* ch, int num) {
    int temp = num;
    char mid[2];
    mid[0] = '0' + temp;
    mid[1] = '\0';
    strcat(ch, mid);
}
```

编码函数，返回值用于检查编码是否成功（可能因为过长而编码失败）

```
int encode(char* in, char* buf);
```

该函数中通过`strcat`函数来进行字符串连接，通过`linkInt`来添加PS，`rand() % 9 + 1`生成非0的一位随机整数；

```
linkInt(in, (rand() % 9 + 1));
```

解码

解码核心是将三位十进制数转化为字符，大致是编码的逆过程，主要区别是程序可以根据`0x00`快速找到message部分，从那部分开始解码；

解码函数

```
int decode(char* plaintext, char* out);
```

编译运行

本程序的编写和编译运行均是在Linux环境(Ubuntu)上进行的，故代码文件夹中仅提供在Linux上运行的`main`文件，理论上源码可以在Windows上正常编译运行，但需要注意是否有`gmp.h`这一第三方库。

Linux下的编译运行命令

```
g++ -c rsa.c -o rsa.o -lgmp
g++ -c main.c -o main.o -lgmp
g++ rsa.o main.o -o main -lgmp
./main
```

运行后程序会在终端依次输出p、q、N、d、e的值，注意为了显示方便，均以16进制的形式输出。

然后程序会等待终端的明文输入，输入支持ASCII码所支持的字符。

然后程序会在终端输出密文和解密后的明文。

验证用例

1.

`RSAES-PKCS1-v1_5 combines the RSAEP and RSADP primitives`

```
p =
0x820635C2F185009137C663219CCA2784EE61508AD307EBA895C65D4BB4D9B1DB6279B422ABF16E
1F2124CBA038CD68097A7EB08A8396279F7A7F9FAF77D2971B43B9132B23F11EF77ECD818FC5BAFE
3E1B20E26A66A67658DA2D1828DE9AEF4F54F1A365858BB7F922C62982AFD2269F0562EAC60E213E
14398E86AC6BA9F3CB

q =
0xE0906A88AA28283148FADAE16942FF780DD9163AB7E1297912B1CA54EAF76B987E7E6D2DC1D0C5
FBDEC7303F6099BE7B54EC4053615F152BDFA1F1495D69EFB7C842B076362C1DACA55EA44EB0B6B93
70E9B8D3C5BF8A6DCA22B4271E762F50E0FB72476B2A4C827A94B21BAC472DC4DD9794653D887672
2E9B229E0EC1BE093F
```

```
N =
0x720ec8a4ccf807f69ee15c2395d103dcefa471cf3595b5d631be37b2932259fd3ad8a25466dd17
c979da8fff4e5824f802ffcbdcc4d84c643b5fe4a56c114acbd6e7917de59146d8f840ea3bc13613
6097e27ad25c3046ab046789a0c352f3fe77d91c840b7f6dd826a757d21fa904a072f5c1154797d8
d9ee7366aa76590766662fa5082184bb53c74377b2bc2da27bcde55c5ab1c4fb7751c675f4568703
3c9bda2856f30fbba1e71302f292483e9d84c0d918b1cf27724dd8009eedf06a396210402900d24f
fee0ccf2847061af47baeed4d693c5d0bade57ab8c6bbefbb65ec53072d8f9fda80b3e02f6213138
e34c854f4b273dfa26df2522ad5616b93d

d =
0x4cb704dd53bfc01b70e1c3476c0d19bc10b35a9934416dfc93988d7dded8cc0e36f65eb8c6370f
38c46ced4d9b344e29425b2a936312d8ced1e0bdb945e69b985f0382285bb04b26c4536538abcbe3
5ca4cc00305d315b56cbaf9832392e224e95edc1f04e8b77265b59530851a32d297e292da552721f
b8f55949447e358373bde7bdf7cf4e1f96f705c0271f28d9e782beaa4f97c370394d970db7e0bc97
c15b718ba82720ee771e227174c583fcc672f574705c6459561c851d856a71b6374bab45df912462
accce3b0ab081e8ad99c1eb86ed865c27ca579317986e734fa05d20733fad66aa5577339222c8edc
b0a483ada044fa93d87def666eea4e195

e = 0x10001

input: RSAES-PKCS1-v1_5 combines the RSAEP and RSADP primitives

ciphertext:
0x21571c4f8a045166156fdd2c409d7d8c7b4001392e180d2d35ab97c69e1532c68906d77295a61b
2494e0be5ff962f4c83ec1933d63778f7f18afffb2baf135fb1c4d768cee65e0f51a2cf142131078
1f71978c2eb5aa2be172d93bde5d0204ef401f260dacf4deb76f9f92a10fbb229d4358e203eb4f48
ea82a2c2ddc40fc127df3c9c05e4b9ff02045a13c63618e0db495cc11a697a38ced7a368ae145320
fff3cf23b7e01c012b2508f9d3c25c77a61452db38b264c305579054467ffd0b489b8387decbe9a3
8c86f5bae24b2e16d43f5f518092e4f31fe876f3f52367071e51df52cfc74e2cdd249c0c628725da
f6ed9ce537b163532439818f4909ada3ff

plaintext: RSAES-PKCS1-v1_5 combines the RSAEP and RSADP primitives
```

2.

数据过长报错

```
p =
0x5671142DF4C5D54B49F929B320332D245F1DE20C6E1CB55BF6B3F274FD6BB29D7F6B432EFAC121
EEB71A3F08C0C74F5361038923F2218F25AACCC868A74271AB18A03C1F0510BE575E9176B0CC9A76
178E858272AD91DF80DB2AF3C24039CC3948C196782C9F2080B03039ABD0B8A93C0D957536A8B42D
D7527E2691128C8F61

q =
0x8BC28893873CD8DC8F1588451AB1A30F224E028D429AA27BCEC9328A877B05788B431FD5F436A4
077C6C5905A20EB14BBD9CA53CC4F0447DC0F17C0104DA367CFEA383B8209AC87BB15AC7B9FA2C28
7E767FB0DBA0856C9B5F879B53B99309AE4FDD70851D8381A734B1047AD38DD7D2B628C00CD10FA8
673B290FBDFE3A48B6

N =
0x2f3115c4220c16870480116b67ccc1d906abed03339b2e52ee17f26c227380eefb60a8fbd9dfaf
e0baae91c2155abff2b480ec40ac9769eece1b83bc6573e67dd7a0b6bac29cd047c45f5823526e60
db74b030fc5daadd5e9b7abe5604358b1319731938fced9df618a0ed02b66a72a94cf962ba450ecc
d0611e4755ac0151b945b1eee790a96d312d0e41db2bf8b244dc9377cd21dd3741cb0fcf3d68211d
2507d6534ee5c1da06fe2425bf5617dd4efefbee1cdbccfb40f2c6017acefc3d43159008c3c56829
14cb473abe976dcfa08856a613a83d80a50d447088f2b506839ccdeabbcf4f56393e2fbc2a21863d
a5afc25dc846b72f5426066fde1aeb398f

d =
0x1f6a328e25081cfaf670f6a7a42ab7246cff8840a85af6447880f27d2f95ea4fac81951c4ad650
736e72854f48fde7677ee78fde21c68ddb4dd31d280df5508b973e4246230556d55f9104d046b079
c3bd7308b906f59675c354bb5df1615f5a7087a153c173c0c2b17477326612888fd150f67aff3bb1
f649a069664ba1793020f35766483b1452f65adb02fceefa800949580157a2198eaf7509e536dd89
a6a7b9ad88df4378861100b3ee67b98ed7b1b0498a77b602728035a906ed6e335940992e8bcb43d53
5a2489960d8989b2e8c7b03191d4c8381298f249bd4a9aeeb8fd43e591cf7f89c33ecb34b1c7efe4
f026df41c5ddd565092e2cbdd94dac14b9

e = 0x10001

input: Set up a self-hosted Sourcegraph instance to search your private
repositories on GitHub, GitLab, Bitbucket and local installations of Git,
Perforce, Subversion and other code repositories.
error: message too long
```

3.

`Set up a self-hosted Sourcegraph instance to search your private repositories on GitHub`

```
p =
0x1BDA199EEA27976E5F204C3ADCD3B24B8158A82673EE392E3299C967F493570551FAA89BDAB8DB
9F29C95418449A2DB8DADEF8248BA80C3616EE045B0F3D4EDE4183F34DEEBCBD8E96B897284F5E80
1464BA50FB05194481DADB4005B42C9AEF7F75ABA4BE3D1D11369911410122493B78A9DFAC416FEF
DE1201C00E13C3C1C9

q =
0x28F89791A724FA3C9EE6B7024BDA8F68837840F851EB79C06F37C909A5BC9D5ABCF4EFDDB96B06
9A8F5EA36A7B44D083110BC0D942A8EFDC2E0090B6637397BBB85933197E6BE243CF6CCF9279B0F6
778944BD1C4ADCD53F5F5F2ACC14AE53F3C7F70F0D0EA5358620A4523BA9448F6DEF5A46CC1819DC
DA92463552BE337814
```

```
N =
0x4751fc7245d91bfc70dbbc31215e80f4c59868653db90822004698edce4f28435fadd15579f3ed
5f67a15d160278d7554cb09488e4f6dcac362f0d6bcfc4ed7239e8302ff5d7156936d3cad8e00fec
076500accf36d5639dee7b971bda818e8802fe2bb5616a240bffcda6dc7b234c6257dc7531e35943
3cc8ab26fa00431a59520cb2bcf900bb21f155eaf21ed9f79bdce80d0d9e2fdd3aa3fa7e50da56af
6ed59b5c43cba78ceb1dd25701b9f77ef9c8ed4e974d14b0b05ca78edb2dc29134a9a4b4fc11deba
38aee86d65a797a8aa8175de568e7bcc96a6d500eb20dbd9d8fcd95190fb49b3ea2a54187acd4d05
5095ef4e9109ab226f3215e20144ca6a9

d = 0x1b9bd757fd46109966af096Set up a self-hosted Sourcegraph instance to search
your private reposSet up a self-hosted Sourcegraph instance to search your
private repositories on GitHubitories on
GitHub9137bdbfbe54a1c8fa4972e58cb368918f5d69ac58a126588eb4ecc433480b8e23e4ebd3d1
e14f54b1127a7415b3057c7b6848b4ebeafed878369a1b63b2d3dc73ae5aeff25d05f15502ac7076
a3c99c08493c6b34c454c7248a5a812a511dd5858aacf74fa823333fec89d48b87dd8f6ca872f785
e0d619f67e748cb1f536dd2619ab9951058a65bb6fe578480d256cade715a34db25268865ed52812
27410233948d084c871aefdc62ab2f8ca912ca3e6500b4a9d6aea4e96b79d01aa7693082d84177a8
5ea4f6299dd0f8e87fab522b0c2a0eded9ef722de1ed922594add62138a4b9e21a9ef3ef4a3126c4
8b510b49087c89

e = 0x10001

input: Set up a self-hosted Sourcegraph instance to search your private
repositories on GitHub

ciphertext:
0x13f080cda2fa5af2aef13e4945cf21d7d68f850ad09575f29e6b22b5255282206ab0ccb42db69
3bc3debd00c94abd19fc14dcf6c554bb2c04c772783e0c74999216c7f473992f584fe08cf03f72ff
78233a60551fcd3c418b99295d115473e5d0d2651029b7dda757c1c0177cd140899132e18ddcacdc
456798fdc1a605417345fa1c0bd6b483e95563f4b2a66e3fc0f372f83bab18df82f7ef3c8a10b0dd
8771a886a4b9b03fffeadf013ed99159640ede57078b5409d6869ec3a51d22d5544efe98d74e1c75
7c63fc8367d3891988477db69c453b2e4a20da48646cca2b1e2620271cd4b6c22e2d523a9e41bef2
1018e05a205f5fe3fc1c134441ca93bee

plaintext: Set up a self-hosted Sourcegraph instance to search your private
repositories on GitHub
```

4.

`This is a C library for RSA encryption. It provides three functions for key generation, encryption, and decryption.`

```
p =
0x7311A845A36038F0965D380C278422F1B7C1EF930FB14D029114AC00361DAE8E6FD96CD4D8E9E3
3D51B0A280D504836C2EA53333A75DE324CB094787AC475B404A8CDD4A53C6EC2D906D9B6FC6999D
771B9F6EFE5AAA2470677727E42D6988C0CF6EAA2AD6ED28F3B4C8DFB19AAAFA7DFFF21A8CC5693F
31DD183F4EC274ECF0

q =
0xD2E293B9C5C531694685ABF41AED8C6C40E7C83C618E6E9CFC53E562F3A5BD2C5C94A77570EC79
33A402301558695F76E32F44F2D4463F794BBEF99057CFB38BECD0F97150D8AC851ECFDB742A6CF0
A0805CF84242F67B6F86CBB1A6462E1F7D7052F781505D8F2FD3C5D8F7B313815BEDBE505882C8B3
ADD8DB407151CD5705
```

```
N =
0x5eca540651c6fa14b86e89a6011844bc17f90c2697fe9d17d65b92963f6fe8e9c2f3a0560e5695
02f2ce39406fd019d3ff4d6ef7b3ffc4be20760c464c18c95ec98a3de82261133022038b8e038039
bf6cdafe23c2f162aaa6d3ae2cae5f13739b8a8445cf905ee78618ab8fc4576e69abe17c0552bd5f
a05dc79b7e215de610a9ba6c4c179f04d5e5317125d0b621071da1c87d097c00df6f89bb786ec494
6fabdc9a176e3fc0aff575d1e247d48257a768c87a2fbd3b23e53529d300ac3564772e79cccc0a2c
db3d7ba4a0f7c77e6369fd600d08fbe331ae2dc7ff800ac88234c629846a66571e1f71ed5839c5d8
a8ef3d74c9066aa6f116ee873185da5491

d =
0x15a339eaf75fe456368385084fd7aa59055fe4562afc04001cfe6a47c1d359e78288fa60aa8539
1ea32e50035582bb629934a3d00756ab227b66770 6d4f8137f7fa1fedfae795a57f26bbcbf7e2d17
018b67b1fb97d4622172537fbed34f00b1b31fbb71f41f908a6921d887c8d9ec6a52c1f5fd811599
6e91f9ae94ec28f30b304b5d5b23b8be7af1aa7cadc5fbf9b612aa4da452cbef42d5eb9c1f9ed0d4
e9f4d6d5d42e2ba0a680db924b5cfb700abcda4e3bcd30ad585df69e69909ebb60a5b08a88afe6b4
4b4dad03e3d256d8837552e37846495ee49a5a96744193461a3d23fb9e72332be727ea556c589495
aaeb3fb6b23609db90dc02718083dda001

e = 0x10001

input: This is a C library for RSA encryption. It provides three functions for
key generation, encryption, and decryption.

ciphertext:
0x1f442e084cbf4148ff4b0b1b2ba7a385f38b5f2642865dfb421fe512fceb1488b3330a2c465459
59aa8c63822bd737d12a2521f9c6a7c7d8e99e15618e4552aa6ebb427fa1e08a1fdf3007e82a5924
8d076b2a4f5c17018c04e1c10ccc634f17cda7089a4ec2c2106ef8fa07d256ab36e6bba75e52def1
526ca55af16ffd1205ecf44ea3472042492932da0e2f14816c25926d61e98fd9b99f61dbc66d3ba7
68e00ff36b0617c519cb3dbe7dd59b09f25887c35110839be8b2e520b536ca4183312deabc5a1a84
d1d02867114d2c58e3cdfe1835debb4a453a8b4cfe5b3d51a4b1a0ee7e3b5e11a8233b9f96cefcfc
432e2beed6cfe66ba99328ebb9a549fd55

plaintext: This is a C library for RSA encryption. It provides three functions
for key generation, encryption, and decryption.
```

5.

`In this section we will explain what happens with your data if you use/sign up on IsThereAnyDeal`

```
p =
0x856F247011CD535B898F699D210189F6D5FF49629A374892AF9A8ACA5EA83CA56221DB46791A9C
193BCB1C04916D7738863CA32861F6FCA886253026F8381607D1ABD248BFC179C3FF834FF0D7A167
7FE0B2836630C2ED9DC32C76C072C6BBEF5ED1869B9C62550F57FEB96E81967222E4D0232954C596
17A0D20ECEAC4B57A2

q =
0x7B8C687B649A1B03346D68E8B351FAD50A800F35814737AD42A23FEC6780ADBC4D3CC3A2EE21C2
DF360C8D4A24FC8592C4EE3DACF19A2B764398E38D9B3C34A9024F02AA8F6918E69AF34A2F91DB18
75CDE46E66317171D024EE53B643713EE2E5581054994529E4CA1DA5A040EF447AAB2B97152E09E7
6A0146EDF28C45A6F1
```

```
N =
0x406595bf3d7b29969f8322b617ec0cad481fffb38343f067c271d198c02089eb6debc4b346d0c6
0b381ddf11d6e4e38e45e52733c4490ec38e34f8c5095a7e367ee498c73ba8f6766645be8e8d21d0
8d16ca04273ee3532e48b7e964d679152552f5ba4b068404b26e782bb6f14e63ae098feaab1782bf
738dc6163ea416b52b341bd909e7b2e5a7d4d26e17f385d625a36e043913160b1cb8d56d15a296d2
680a944baad3ab3c00ccaf56f5705789fbbeea4585702db0984fdb615759298811438afdc6742528
5423293c50e62e67e4e9525b4466b96eb6d6f0eb93d1ca43080184da1e96c88c0a5248974f5e8e75
014272e39ba5f44b4244d65c23929070f3

d =
0xccda5b63e1d0ca9bbd130c78f7e090f32b5d9faf0f0430dfe24ef0d35b5c2762286abb7112f551
0c9f9766e6f915684e4f4a894ac43f241833698f763e2b01159ad4d8adf1bbad4d384ae5c5d4709d
5e3600aef5a9b8be039e906e85b93fb564e6638c1cdebd59b3c477c39854496a0f599bd8882c6261
66d6ba2e4956f4659bb8c8255d324b3f4be2b7e20d13f09d994de2b1577e493614dc2ef15793208d
daaa8c34e39a6d91f5e205f90348b3c7513ae4bf08fdfb76aa1f59587f553117ee8ef7a8e705db9f
fd903db2754c3725fa807b6f27bc390c46967fd717217e56f79d47948f15ac6e550729a448e4156c
92fa46d7eed0bd634a4a68e1e59d26591

e = 0x10001

input: In this section we will explain what happens with your data if you
use/sign up on IsThereAnyDeal

ciphertext:
0x204ebcf32e5503655294c6b9c9171c4e55483f5425c4ae31e948e9d7b9e52e09c4434723e5fb5d
a221f75ffde3d5e22ddecb28d1bb149a93a15d5afe2c2c926b1342944fd5b4c5f4b7df7fb9d9f192
077f83c6bfbd72af06f12709eadb51f9196a995c5b38d1ffb4af24f5a74409ad1e2439e5cf129f3c
b0a61f5265f213abb9bfd9095e83c83612820d37ead675429226cf1bdab44d8c5dea0b3d51642008
e583aab9039b33acccf273ea79ff929536b22b74ce23d3a9f9ccfd7ee98785e8c285d63f442502e2
e95050b805c1d37cadb2933805a90d32d8460a169fe3956d903226e61d7bb9ef59bc5a1d6e3cebce
5ec1b91d869c85cfe13035d88fa59e97ac

plaintext: In this section we will explain what happens with your data if you
use/sign up on IsThereAnyDeal
```

源代码

rsa.h

```c
#ifndef _RSA_H_
#define _RSA_H_

#include "gmp.h"

#define KEY_LENGTH 2048
#define BASE 16  //输入输出的数字进制

struct triple {
    char x[KEY_LENGTH + 10];
    char y[KEY_LENGTH + 10];
    char d[KEY_LENGTH + 10];
};

struct keyPair {
    char *N;
```

```
    char *d;
    int e;
};

struct keyPair *rsaGenKey();

char *rsaEncrypt(const char *plaintext, const char *key_n, int key_e);

char *rsaDecrypt(const char *ciphertext, const char *key_n, const char *key_d);

int encode(char *in, char *buf);

int decode(char *plaintext, char *out);

#endif  // !_RSA_H_
```

rsa.c

```
#include "rsa.h"

#include <gmp.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

struct triple extEuclid(char* a, char* b) {
    char x[KEY_LENGTH + 10];
    char y[KEY_LENGTH + 10];
    char d[KEY_LENGTH + 10];
    struct triple ee;
    ee.x[0] = '1';
    ee.x[1] = '\0';
    ee.y[0] = '0';
    ee.y[1] = '\0';
    strcpy(ee.d, a);
    if (strcmp(b, "0") == 0) {
        return ee;
    }

    mpz_t amb, ma, mb;
    mpz_init(amb);
    mpz_init_set_str(ma, a, BASE);
    mpz_init_set_str(mb, b, BASE);
    mpz_mod(amb, ma, mb);
    char mid[KEY_LENGTH + 10];
    mpz_get_str(mid, BASE, amb);

    ee = extEuclid(b, mid);
    strcpy(x, ee.y);

    mpz_div(ma, ma, mb);
    mpz_t ex, ey, my;
    mpz_init(my);
    mpz_init_set_str(ex, ee.x, BASE);
    mpz_init_set_str(ey, ee.y, BASE);
    mpz_mul(ma, ma, ey);
```

```
    mpz_sub(my, ex, ma);
    char temp[KEY_LENGTH + 10];
    mpz_get_str(temp, BASE, my);
    // y = ee.x - (a / b) * ee.y;

    struct triple ref;
    strcpy(ref.x, x);
    strcpy(ref.y, temp);
    strcpy(ref.d, ee.d);

    mpz_clear(amb);
    mpz_clear(ma);
    mpz_clear(mb);
    mpz_clear(ex);
    mpz_clear(ey);
    mpz_clear(my);
    return ref;
}

// 生成两个大素数
mpz_t* rsaGenPrime() {
    // 随机数
    gmp_randstate_t grt;
    gmp_randinit_default(grt);
    gmp_randseed_ui(grt, time(NULL));

    mpz_t p, q;
    mpz_init(p);
    mpz_init(q);

    // 随机生成大整数
    mpz_urandomb(p, grt, KEY_LENGTH / 2);
    mpz_urandomb(q, grt, KEY_LENGTH / 2);

    mpz_t* ref = (mpz_t*)malloc(sizeof(mpz_t) * 2);
    mpz_init(ref[0]);
    mpz_init(ref[1]);

    // 使用GMP自带的素数生成函数
    mpz_nextprime(ref[0], p);
    mpz_nextprime(ref[1], q);

    gmp_printf("p = 0x%ZX\n\n", p);
    gmp_printf("q = 0x%ZX\n\n", q);

    mpz_clear(p);
    mpz_clear(q);

    return ref;
}

// 生成密钥对
struct keyPair* rsaGenKey() {
    mpz_t* PandQ = rsaGenPrime();

    mpz_t N, e, phiN;
    mpz_init(N);
```

```
    mpz_init(phiN);
    // 设置e为65537
    mpz_init_set_ui(e, 65537);

    // 计算n=p*q
    mpz_mul(N, PandQ[0], PandQ[1]);

    // 计算欧拉函数ϕ(N)=(p-1)*(q-1)
    mpz_sub_ui(PandQ[0], PandQ[0], 1);
    mpz_sub_ui(PandQ[1], PandQ[1], 1);
    mpz_mul(phiN, PandQ[0], PandQ[1]);

    // 计算数论倒数
    mpz_t d;
    mpz_init(d);
    // mpz_invert(d, e, phiN);
    char ce[KEY_LENGTH + 10];
    char cphiN[KEY_LENGTH + 10];
    mpz_get_str(ce, BASE, e);
    mpz_get_str(cphiN, BASE, phiN);
    struct triple ee = extEuclid(cphiN, ce);
    mpz_set_str(d, ee.y, BASE);

    mpz_t ze;
    mpz_init(ze);
    for (; mpz_cmp(d, ze) < 0; mpz_add(d, d, phiN)) {
    }

    struct keyPair* ref = (struct keyPair*)malloc(sizeof(struct keyPair));

    char* buf_n = (char*)malloc(sizeof(char) * (KEY_LENGTH + 10));
    char* buf_d = (char*)malloc(sizeof(char) * (KEY_LENGTH + 10));

    mpz_get_str(buf_n, BASE, N);
    ref->N = buf_n;
    mpz_get_str(buf_d, BASE, d);
    ref->d = buf_d;
    ref->e = 65537;

    mpz_clear(PandQ[0]);
    mpz_clear(PandQ[1]);
    mpz_clear(N);
    mpz_clear(d);
    mpz_clear(e);
    mpz_clear(phiN);
    free(PandQ);

    return ref;
}

// 加密函数
char* rsaEncrypt(const char* plaintext, const char* key_n, int key_e) {
    mpz_t M, C, N;
    mpz_init_set_str(M, plaintext, BASE);
    mpz_init_set_str(N, key_n, BASE);
    mpz_init_set_ui(C, 0);

    mpz_powm_ui(C, M, key_e, N);  //使用GMP中模幂计算函数
```

```
    char* ref = (char*)malloc(sizeof(char) * (KEY_LENGTH + 10));
    mpz_get_str(ref, BASE, C);

    return ref;
}

// 解密函数
char* rsaDecrypt(const char* ciphertext, const char* key_n, const char* key_d) {
    mpz_t M, C, N, d;
    mpz_init_set_str(C, ciphertext, BASE);
    mpz_init_set_str(N, key_n, BASE);
    mpz_init_set_str(d, key_d, BASE);
    mpz_init(M);

    // 模幂计算
    mpz_powm(M, C, d, N);

    char* ref = (char*)malloc(sizeof(char) * (KEY_LENGTH + 10));
    mpz_get_str(ref, BASE, M);

    return ref;
}

void linkInt(char* ch, int num) {
    int temp = num;
    char mid[2];
    mid[0] = '0' + temp;
    mid[1] = '\0';
    strcat(ch, mid);
}

int encode(char* in, char* buf) {
    if (strlen(buf) > (KEY_LENGTH / 4 - 9 - 3 * 8) / 3) {
        printf("error: message too long\n");
        return 1;
    }

    in[0] = '1';
    in[1] = '\0';

    strcat(in, "00");
    strcat(in, "002");

    for (int i = 0; i < KEY_LENGTH / 4 - strlen(buf) * 4 - 9; i++) {
        linkInt(in, (rand() % 9 + 1));
    }
    strcat(in, "000");

    char add[KEY_LENGTH + 10];
    add[0] = '\0';
    for (int i = 0; buf[i] != '\0'; i++) {
        int temp = (int)buf[i];
        char mid[4];
        mid[0] = '0' + temp / 100;
        mid[1] = '0' + (temp % 100) / 10;
        mid[2] = '0' + (temp % 10);
        mid[3] = '\0';
```

```
        strcat(add, mid);
    }

    strcat(in, add);

    return 0;
}

int decode(char* plaintext, char* out) {
    plaintext[0] = '\0';
    int i = 0;
    for (i = 6; out[i] != '0'; i++) {
    }
    i += 3;

    for (; out[i] != '\0'; i += 3) {
        int temp = 0;
        for (int j = 0; j < 3; j++) {
            temp *= 10;
            temp += (int)(out[i + j] - '0');
        }
        char mid[2];
        mid[0] = (char)(temp);
        mid[1] = '\0';
        strcat(plaintext, mid);
    }

    return 0;
}
```

main.c

```
#include "gmp.h"
#include "rsa.h"
#include "stdio.h"
#include "stdlib.h"
#include "string.h"

int main(int argc, char const* argv[]) {
    struct keyPair* p = rsaGenKey();

    printf("N = 0x%s\n\n", p->N);
    printf("d = 0x%s\n\n", p->d);
    printf("e = 0x%x\n\n", p->e);

    char buf[KEY_LENGTH + 10];
    char in[KEY_LENGTH + 10];

    // 整行输入
    printf("input: ");
    scanf("%[^\n]", buf);
    getchar();

    int error = encode(in, buf);
    if (error == 1) {
        return 0;
    }
```

```
    char* ciphertext = rsaEncrypt(in, p->N, p->e);
    printf("\nciphertext: 0x%s\n\n", ciphertext);

    char* out = rsaDecrypt(ciphertext, p->N, p->d);

    char plaintext[KEY_LENGTH + 10];
    error = decode(plaintext, out);
    if (error == 1) {
        return 0;
    }

    printf("plaintext: %s\n\n", plaintext);

    free(ciphertext);
    free(out);

    return 0;
}
```