网络与信息安全第3次作业：实现一个RSA算法

1. 算法思想

- 使用快速幂算法实现大素数模n乘法
- 使用Fermat素性测试生成两个指定位数的随机大素数p,q。
- 计算n=pq,phi(n)=(p-1)(q-1)
- 取一个与phi(n)互素的数e（直接取e=65537）
- 使用拓展欧几里得算法求e模phi(n)的模反元素d
- 给出密钥对{e,n}，{d,n}

2. 程序

- 详见文件夹中附带代码或后附代码
- src文件下为实现的代码，各个函数声明处有充分注释
- test文件夹下为单元测试
- main.cpp是一个样例
- 使用cmake构建
- 运行：

```
cmake -S . -B build
cmake --build build
(optional) ./build/unit_test (运行单元测试)
./build/main
```

3.结果

使用1024位整数进行计算，中间结果和结果输出如下

```
tang@LAPTOP-G3IKMRA5:~/workspace/myrsh$ ./build/main
p=1653079719030224946745691449078359496425176539605196949301259155281133540788035178399334064111325677835726349616845146304134570235195859433485493616885935537454329176399805150273530249427867061673047373793797102471201850854766414211451678497156612244437235190499509806095521409000487834771244428995942088233373
q=1175204021908664165049226434311098225369988352075174985639380940741695283722068195149421977177339047508127196214130640120455540653339661193023467530009402881392474462890382843648148361569534407023120994533931167970970805381998599351993828766541070124124544922132764036506567246437670820811580293717874554865 59
n=1942705934339964880602456636578898091538382384646507303813296512499607235346826255481499392922079024578048762630714322891792963363742520174902499008934179659966117492774302539050328103930084668100603313800445880067330212664817599090038507399593609081269186559475622286161473615035566404687732938040517498839492147864689931497703405414825967992409773012443521777262261320208734113020121254015097427079133728889448436265404422473027845208165060753636277519974586338986937543365864113620334011493026414595379882870065261817344096347678787222499191838887684258238440227218456633896669082058387928450505578105083660 6543507
phi(n)=1942705934339964880602456636578898091538382384646507303813296512499607235346826255481499392922079024578048762630714322891792963363742520174902499008934179659966117492774302539050328103930084668100603313800445880067330212664817599090038507399593609081269186559475622286161473615035566404687732938040517498839463865027280542606585456235992073415191821363526718057912854919248505824775020220279609866666247081636009900807094664608781944099279705547371187908505632954798469506972962233681117225383052399908418199186787979112922369785311137086864736766250707434552822426092133895470648195504006341894677330 8779454942233576
key1={65537,19427059343399648806024566365788980915383823846465073038132965124996072353468262554814993929220790245780487626307143228917929633637425201749024990089341796599661174927743025390503281039300846681006033138004458800673302126648175990900385073995936090812691865594756222861614736150355664046877329380405174988394921478646899314977034054148259679924097730124435217772622613202087341130201212540150974270791337288894484362654044224730278452081650607536362775199745863389869375433658641136203340114930264145953798828700652618173440963476787872224991918388876842582384402272184566338966690820583879284505055781050836606543507}
key2={10010106763618455842205831539532296207205034522045242403904849158582058490688767539154180539174159113321654739349770659281467961583566712511448874533881366699329511819811010946669900932544216115642960969181875417549426118296264019358760754440526204291526795692041486333122785984426513560263676027997960757787182089217729930158808653376446332172301541972463454552033080209363076307555679664711864380037611990137848595504032795088874600651323126879347184717385098379631279549105079828063177683439934158186577684108000666290252460522760757866340163508545098789346693631180940072266785463386709647366241341325012457000 2985,194270593433996488060245663657889809153838238464650730381329651249960723534682625548149939292207902457804876263071432289179296336374252017490249900893417965996611749277430253905032810393008466810060331380044588006733021266481759909003850739959360908126918655947562228616147361503556640468773293804051749883949214786468993149770340541482596799240977301244352177726226132020873411302012125401509742707913372888944843626540442247302784520816506075363627751997458633898693754336586411362033401149302641459537988287006526181734409634767878722249919183888768425823844022721845663389666908205838792845050557810508366 06543507}
raw content: 14174550714229486541878239327136436563046920429330779387088270007432996229576536548010888407866294807297663420247722748101802587087398503847355539801683954657245425946617140018501213797726254035872944727411380440475663452923757410169503125484668445780505753079725391214311098702354011609544391566206108874 7363
encrypted content 73504928073079927478734931304380100479044041535278710427019996443975410881606246991271976071926646096761636909544054944306132798656186847762143533890140312642337336344985069587513163097652799784799327214104720372866101142380389993416258003423252781928524051016014903909542609537023288986465746739728702835341270431695979796762283998405686593173148434308170156258454261792169621882395950292085669901012603419321649473749171001948482422772235897062002320315509664292408150967356181793217683799312562138470742976399111675967002982341646228093691261931940982609044811063288436137490017440301436 194806177299807931966638 16
decrypted content 14174550714229486541878239327136436563046920429330779387088270007432996229576536548010888407866294807297663420247722748101802587087398503847355539801683954657245425946617140018501213797726254035872944727411380440475663452923757410169503125484668445780505753079725391214311098702354011609544391566206108874 7363
```

单元测试结果

```
tang@LAPTOP-G3IKMRA5:~/workspace/myrsh$ ./build/unit_test
Running main() from /home/tang/googletest/googletest/src/gtest_main.cc
[==========] Running 7 tests from 4 test suites.
[----------] Global test environment set-up.
[----------] 2 tests from TestExtendedEuclidean
[ RUN      ] TestExtendedEuclidean.TestExtendedEuclidean1
[       OK ] TestExtendedEuclidean.TestExtendedEuclidean1 (0 ms)
[ RUN      ] TestExtendedEuclidean.TestExtendedEuclidean2
[       OK ] TestExtendedEuclidean.TestExtendedEuclidean2 (2 ms)
[----------] 2 tests from TestExtendedEuclidean (3 ms total)

[----------] 2 tests from TestMyRsaKey
[ RUN      ] TestMyRsaKey.TestGenerateKeyPair
[       OK ] TestMyRsaKey.TestGenerateKeyPair (66 ms)
[ RUN      ] TestMyRsaKey.TestEncodeAndDecode
[       OK ] TestMyRsaKey.TestEncodeAndDecode (1781 ms)
[----------] 2 tests from TestMyRsaKey (1848 ms total)

[----------] 2 tests from TestPowerMod
[ RUN      ] TestPowerMod.TestPowerMod1
[       OK ] TestPowerMod.TestPowerMod1 (0 ms)
[ RUN      ] TestPowerMod.TestPowerMod2
[       OK ] TestPowerMod.TestPowerMod2 (0 ms)
[----------] 2 tests from TestPowerMod (0 ms total)

[----------] 1 test from TestPowerPrime
[ RUN      ] TestPowerPrime.TestPowerPrime1
[       OK ] TestPowerPrime.TestPowerPrime1 (57 ms)
[----------] 1 test from TestPowerPrime (58 ms total)

[----------] Global test environment tear-down
[==========] 7 tests from 4 test suites ran. (1910 ms total)
[  PASSED  ] 7 tests.
```

4. 后附代码

src/util.h

```
#ifndef __UTIL_H_
#define __UTIL_H_
#include<vector>
#include <boost/multiprecision/cpp_int.hpp>
/**
 * @brief 生成一个指定位数的大整数，保证最高位不是0
 * @param bits 位数
*/
boost::multiprecision::cpp_int generateRandomBigNumberWithBits(int bits);

/**
 * @brief 使用 Fermat素性测试对n是否为素数进行测试
*/
bool primeTest(boost::multiprecision::cpp_int n);

/**
 * @brief 使用快速幂算法计算a的n次幂对modbase取模
 * @param a 底数
```

```
 * @param power 指数
 * @param modBase 模数
 * @return a^n mod modbase
*/
boost::multiprecision::cpp_int powermod(boost::multiprecision::cpp_int
a,boost::multiprecision::cpp_int power,boost::multiprecision::cpp_int modBase);

/**
 * @brief 生成一个指定位数的大素数，保证最高位不是0
 * @param bits 位数
*/
boost::multiprecision::cpp_int generateRandomPrimeNumberWithBits(int bits);


/**
 * @brief 拓展欧几里得算法，求解不定方程ax+by=gcd(a,b)的一个特解
 * @param a 入参，即不定方程中的a
 * @param b 入参，即不定方程中的b
 * @param x 出参，引用类型，函数返回时会被修改为不定方程里x的值
 * @param x 出参，引用类型，函数返回时会被修改为不定方程里y的值
 * @return gcd(a,b)
*/
boost::multiprecision::cpp_int extendedEuclidean(
    boost::multiprecision::cpp_int a,
    boost::multiprecision::cpp_int b,
    boost::multiprecision::cpp_int &x,
    boost::multiprecision::cpp_int &y
);
#endif
```

src/util.cpp

```
#include "util.h"
using namespace std;
using boost::multiprecision::cpp_int;
// primes smaller than 100
vector<int> primes = {
    2,    3,    5,    7,    11,   13,   17,   19,   23,   29,   31,   37,   41,
  43,   47,   53,   59,   61,   67,
    71,   73,   79,   83,   89,   97,   101,  103,  107,  109,  113,  127,  131,
 137,  139,  149,  151,  157,  163,
    167,  173,  179,  181,  191,  193,  197,  199,  211,  223,  227,  229,  233,
 239,  241,  251,  257,  263,  269,
    271,  277,  281,  283,  293,  307,  311,  313,  317,  331,  337,  347,  349,
 353,  359,  367,  373,  379,  383,
    389,  397,  401,  409,  419,  421,  431,  433,  439,  443,  449,  457,  461,
 463,  467,  479,  487,  491,  499,
    503,  509,  521,  523,  541,  547,  557,  563,  569,  571,  577,  587,  593,
 599,  601,  607,  613,  617,  619,
    631,  641,  643,  647,  653,  659,  661,  673,  677,  683,  691,  701,  709,
 719,  727,  733,  739,  743,  751,
    757,  761,  769,  773,  787,  797,  809,  811,  821,  823,  827,  829,  839,
 853,  857,  859,  863,  877,  881,
    883,  887,  907,  911,  919,  929,  937,  941,  947,  953,  967,  971,  977,
 983,  991,  997,  1009, 1013, 1019,
```

```
    1021, 1031, 1033, 1039, 1049, 1051, 1061, 1063, 1069, 1087, 1091, 1093,
1097, 1103, 1109, 1117, 1123, 1129, 1151,
    1153, 1163, 1171, 1181, 1187, 1193, 1201, 1213, 1217, 1223, 1229, 1231,
1237, 1249, 1259, 1277, 1279, 1283, 1289,
    1291, 1297, 1301, 1303, 1307, 1319, 1321, 1327, 1361, 1367, 1373, 1381,
1399, 1409, 1423, 1427, 1429, 1433, 1439,
    1447, 1451, 1453, 1459, 1471, 1481, 1483, 1487, 1489, 1493, 1499, 1511,
1523, 1531, 1543, 1549, 1553, 1559, 1567,
    1571, 1579, 1583, 1597, 1601, 1607, 1609, 1613, 1619, 1621, 1627, 1637,
1657, 1663, 1667, 1669, 1693, 1697, 1699,
    1709, 1721, 1723, 1733, 1741, 1747, 1753, 1759, 1777, 1783, 1787, 1789,
1801, 1811, 1823, 1831, 1847, 1861, 1867,
    1871, 1873, 1877, 1879, 1889, 1901, 1907, 1913, 1931, 1933, 1949, 1951,
1973, 1979, 1987, 1993, 1997, 1999, 2003,
    2011, 2017, 2027, 2029, 2039, 2053, 2063, 2069, 2081, 2083, 2087, 2089,
2099, 2111, 2113, 2129, 2131, 2137, 2141,
    2143, 2153, 2161, 2179, 2203, 2207, 2213, 2221, 2237, 2239, 2243, 2251,
2267, 2269, 2273, 2281, 2287, 2293, 2297,
    2309, 2311, 2333, 2339, 2341, 2347, 2351, 2357, 2371, 2377, 2381, 2383,
2389, 2393, 2399, 2411, 2417, 2423, 2437,
    2441, 2447, 2459, 2467, 2473, 2477, 2503, 2521, 2531, 2539, 2543, 2549,
2551, 2557, 2579, 2591, 2593, 2609, 2617,
    2621, 2633, 2647, 2657, 2659, 2663, 2671, 2677, 2683, 2687, 2689, 2693,
2699, 2707, 2711, 2713, 2719, 2729, 2731,
    2741, 2749, 2753, 2767, 2777, 2789, 2791, 2797, 2801, 2803, 2819, 2833,
2837, 2843, 2851, 2857, 2861, 2879, 2887,
    2897, 2903, 2909, 2917, 2927, 2939, 2953, 2957, 2963, 2969, 2971, 2999,
3001, 3011, 3019, 3023, 3037, 3041, 3049,
    3061, 3067, 3079, 3083, 3089, 3109, 3119, 3121, 3137, 3163, 3167, 3169,
3181, 3187, 3191, 3203, 3209, 3217, 3221,
    3229, 3251, 3253, 3257, 3259, 3271, 3299, 3301, 3307, 3313, 3319, 3323,
3329, 3331, 3343, 3347, 3359, 3361, 3371,
    3373, 3389, 3391, 3407, 3413, 3433, 3449, 3457, 3461, 3463, 3467, 3469,
3491, 3499, 3511, 3517, 3527, 3529, 3533,
    3539, 3541, 3547, 3557, 3559, 3571, 3581, 3583, 3593, 3607, 3613, 3617,
3623, 3631, 3637, 3643, 3659, 3671, 3673,
    3677, 3691, 3697, 3701, 3709, 3719, 3727, 3733, 3739, 3761, 3767, 3769,
3779, 3793, 3797, 3803, 3821, 3823, 3833,
    3847, 3851, 3853, 3863, 3877, 3881, 3889, 3907, 3911, 3917, 3919, 3923,
3929, 3931, 3943, 3947, 3967, 3989, 4001,
    4003, 4007, 4013, 4019, 4021, 4027, 4049, 4051, 4057, 4073, 4079, 4091,
4093, 4099, 4111, 4127, 4129, 4133, 4139,
    4153, 4157, 4159, 4177, 4201, 4211, 4217, 4219, 4229, 4231, 4241, 4243,
4253, 4259, 4261, 4271, 4273, 4283, 4289,
    4297, 4327, 4337, 4339, 4349, 4357, 4363, 4373, 4391, 4397, 4409, 4421,
4423, 4441, 4447, 4451, 4457, 4463, 4481,
    4483, 4493, 4507, 4513, 4517, 4519, 4523, 4547, 4549, 4561, 4567, 4583,
4591, 4597, 4603, 4621, 4637, 4639, 4643,
    4649, 4651, 4657, 4663, 4673, 4679, 4691, 4703, 4721, 4723, 4729, 4733,
4751, 4759, 4783, 4787, 4789, 4793, 4799,
    4801, 4813, 4817, 4831, 4861, 4871, 4877, 4889, 4903, 4909, 4919, 4931,
4933, 4937, 4943, 4951, 4957, 4967, 4969,
    4973, 4987, 4993, 4999, 5003, 5009, 5011, 5021, 5023, 5039, 5051, 5059,
5077, 5081, 5087, 5099, 5101, 5107, 5113,
    5119, 5147, 5153, 5167, 5171, 5179, 5189, 5197, 5209, 5227, 5231, 5233,
5237, 5261, 5273, 5279, 5281, 5297, 5303,
    5309, 5323, 5333, 5347, 5351, 5381, 5387, 5393, 5399, 5407, 5413, 5417,
5419, 5431, 5437, 5441, 5443, 5449, 5471,
```

```
    5477, 5479, 5483, 5501, 5503, 5507, 5519, 5521, 5527, 5531, 5557, 5563,
5569, 5573, 5581, 5591, 5623, 5639, 5641,
    5647, 5651, 5653, 5657, 5659, 5669, 5683, 5689, 5693, 5701, 5711, 5717,
5737, 5741, 5743, 5749, 5779, 5783, 5791,
    5801, 5807, 5813, 5821, 5827, 5839, 5843, 5849, 5851, 5857, 5861, 5867,
5869, 5879, 5881, 5897, 5903, 5923, 5927,
    5939, 5953, 5981, 5987, 6007, 6011, 6029, 6037, 6043, 6047, 6053, 6067,
6073, 6079, 6089, 6091, 6101, 6113, 6121,
    6131, 6133, 6143, 6151, 6163, 6173, 6197, 6199, 6203, 6211, 6217, 6221,
6229, 6247, 6257, 6263, 6269, 6271, 6277,
    6287, 6299, 6301, 6311, 6317, 6323, 6329, 6337, 6343, 6353, 6359, 6361,
6367, 6373, 6379, 6389, 6397, 6421, 6427,
    6449, 6451, 6469, 6473, 6481, 6491, 6521, 6529, 6547, 6551, 6553, 6563,
6569, 6571, 6577, 6581, 6599, 6607, 6619,
    6637, 6653, 6659, 6661, 6673, 6679, 6689, 6691, 6701, 6703, 6709, 6719,
6733, 6737, 6761, 6763, 6779, 6781, 6791,
    6793, 6803, 6823, 6827, 6829, 6833, 6841, 6857, 6863, 6869, 6871, 6883,
6899, 6907, 6911, 6917, 6947, 6949, 6959,
    6961, 6967, 6971, 6977, 6983, 6991, 6997, 7001, 7013, 7019, 7027, 7039,
7043, 7057, 7069, 7079, 7103, 7109, 7121,
    7127, 7129, 7151, 7159, 7177, 7187, 7193, 7207, 7211, 7213, 7219, 7229,
7237, 7243, 7247, 7253, 7283, 7297, 7307,
    7309, 7321, 7331, 7333, 7349, 7351, 7369, 7393, 7411, 7417, 7433, 7451,
7457, 7459, 7477, 7481, 7487, 7489, 7499,
    7507, 7517, 7523, 7529, 7537, 7541, 7547, 7549, 7559, 7561, 7573, 7577,
7583, 7589, 7591, 7603, 7607, 7621, 7639,
    7643, 7649, 7669, 7673, 7681, 7687, 7691, 7699, 7703, 7717, 7723, 7727,
7741, 7753, 7757, 7759, 7789, 7793, 7817,
    7823, 7829, 7841, 7853, 7867, 7873, 7877, 7879, 7883, 7901, 7907, 7919,
7927, 7933, 7937, 7949, 7951, 7963, 7993,
    8009, 8011, 8017, 8039, 8053, 8059, 8069, 8081, 8087, 8089, 8093, 8101,
8111, 8117, 8123, 8147, 8161, 8167, 8171,
    8179, 8191, 8209, 8219, 8221, 8231, 8233, 8237, 8243, 8263, 8269, 8273,
8287, 8291, 8293, 8297, 8311, 8317, 8329,
    8353, 8363, 8369, 8377, 8387, 8389, 8419, 8423, 8429, 8431, 8443, 8447,
8461, 8467, 8501, 8513, 8521, 8527, 8537,
    8539, 8543, 8563, 8573, 8581, 8597, 8599, 8609, 8623, 8627, 8629, 8641,
8647, 8663, 8669, 8677, 8681, 8689, 8693,
    8699, 8707, 8713, 8719, 8731, 8737, 8741, 8747, 8753, 8761, 8779, 8783,
8803, 8807, 8819, 8821, 8831, 8837, 8839,
    8849, 8861, 8863, 8867, 8887, 8893, 8923, 8929, 8933, 8941, 8951, 8963,
8969, 8971, 8999, 9001, 9007, 9011, 9013,
    9029, 9041, 9043, 9049, 9059, 9067, 9091, 9103, 9109, 9127, 9133, 9137,
9151, 9157, 9161, 9173, 9181, 9187, 9199,
    9203, 9209, 9221, 9227, 9239, 9241, 9257, 9277, 9281, 9283, 9293, 9311,
9319, 9323, 9337, 9341, 9343, 9349, 9371,
    9377, 9391, 9397, 9403, 9413, 9419, 9421, 9431, 9433, 9437, 9439, 9461,
9463, 9467, 9473, 9479, 9491, 9497, 9511,
    9521, 9533, 9539, 9547, 9551, 9587, 9601, 9613, 9619, 9623, 9629, 9631,
9643, 9649, 9661, 9677, 9679, 9689, 9697,
    9719, 9721, 9733, 9739, 9743, 9749, 9767, 9769, 9781, 9787, 9791, 9803,
9811, 9817, 9829, 9833, 9839, 9851, 9857,
    9859, 9871, 9883, 9887, 9901, 9907, 9923, 9929, 9931, 9941, 9949, 9967,
9973};
cpp_int generateRandomBigNumberWithBits(int bits) {
    cpp_int mask = 1;
    cpp_int res = 0;
    int times = bits / 32;
```

```
    if (bits % 32 != 0) {
        times++;
    }
    uint32_t lastMask=0xffffffff;
    if(bits%32!=0){
        lastMask=1;
        for(int i=1;i<bits%32;i++){
            lastMask=lastMask&(lastMask<<1);
        }
    }

    for (int i = 0; i < times; i++) {
        cpp_int tmp = rand();
        if(i==times-1){
            tmp=tmp&lastMask;
        }
        res |= (tmp << (i * 32));
    }
    if (!(res & (mask << (bits - 1)))) {
        // ensure that the 1st bit is not zero
        res |= (mask << (bits - 1));
    }
    return res;
}
bool primeTest(cpp_int n) {
    //首先排除掉偶数和10000以内的质数，节省时间
    if (n % 2 == 0) {
        return false;
    }
    for (auto i : primes) {
        if(n%i==0){
            return false;
        }
    }
    //费马素性测试
    for(int i=0;i<5;i++){
        cpp_int tmp=((rand())%100)+3;
        cpp_int f=powermod(tmp,n-1,n);
        if(f!=1){
            return false;
        }

    }
    return true;
}

cpp_int powermod(cpp_int a,cpp_int power,cpp_int modBase){
    cpp_int mask=1;
    cpp_int current=a;
    cpp_int res=1;
    if(power==0){return 1;}
    //快速幂算法
    while(1){
        if(mask>power){
            break;
        }
        if(mask&power){
            res=((res%modBase)*(current%modBase))%modBase;
```

```
        }
        mask=(mask<<1);
        current=((current%modBase)*(current%modBase))%modBase;
    }
    return res%modBase;
}

cpp_int generateRandomPrimeNumberWithBits(int bits){
    cpp_int tmp=generateRandomBigNumberWithBits(bits);
    if(tmp%2==0){
        tmp+=1;
    }
    while(1){
        if(primeTest(tmp)){
            return tmp;
        }else{
            tmp+=2;
        }
    }
}



cpp_int extendedEuclidean(cpp_int a,cpp_int b,cpp_int &x,cpp_int &y){
    if(b==0){
        x=1;
        y=0;
        return a;
    }
    cpp_int gcd=extendedEuclidean(b,a%b,y,x);
    y-=a/b*x;
    return gcd;

}
```

src/MyRsaKey.h

```
#ifndef __MY_RSA_H_
#define __MY_RSA_H_
#include<map>
#include <boost/multiprecision/cpp_int.hpp>
#include"util.h"
class MyRsaKey{
    public:
    /**
     * @brief 构造函数
     * @param _e 对应公钥{e,n}里的e或者私钥{d,n}里的d(反正公钥私钥地位等同)
     * @param _n 对应公钥{e,n}里的n或者私钥{d,n}里的n(反正公钥私钥地位等同)
     *
    */
    MyRsaKey(boost::multiprecision::cpp_int _e,boost::multiprecision::cpp_int
_n):n(_n),e(_e){}
    /**
     * @brief 复制构造函数
    */
    MyRsaKey(const MyRsaKey& )=default;
```

```
    /**
     * @brief 生成一个公钥-私钥对
     * @param bits 位数
     * @return pair<MyRsaKey,MyRsaKey>, 两个分别是公钥和私钥
    */
    static std::pair<MyRsaKey,MyRsaKey>generateKeyPair(int bits);

    /**
     * @brief 在已经生成了两个大素数情况下，用给定大素数生成一个公钥-私钥对
     * @param p 大素数
     * @param q 大素数
     * @return pair<MyRsaKey,MyRsaKey>, 两个分别是公钥和私钥
    */
    static
std::pair<MyRsaKey,MyRsaKey>generateKeyPair(boost::multiprecision::cpp_int
p,boost::multiprecision::cpp_int q);

    /**
     * @brief 使用密钥加密
     * @param content 明文
     * @return 密文
     *
    */
    boost::multiprecision::cpp_int encode(boost::multiprecision::cpp_int
content);

    /**
     * @brief 使用密钥解密
     * @param content 密文
     * @return 明文
     *
    */
    boost::multiprecision::cpp_int decode(boost::multiprecision::cpp_int
content);

    boost::multiprecision::cpp_int getE(){return e;}
    boost::multiprecision::cpp_int getN(){return n;}

    private:


    //对应公钥{e,n}里的e或者私钥{d,n}里的d(反正公钥私钥地位等同)
    boost::multiprecision::cpp_int n;
    //对应公钥{e,n}里的n或者私钥{d,n}里的n(反正公钥私钥地位等同)
    boost::multiprecision::cpp_int e;
};
#endif
```

src/MyRsaKey.cpp

```
#include "MyRsaKey.h"
using namespace std;
using boost::multiprecision::cpp_int;
std::pair<MyRsaKey, MyRsaKey> MyRsaKey::generateKeyPair(int bits) {
    while (1) {
        //生成两个大素数
```

```
        cpp_int p = generateRandomPrimeNumberWithBits(bits);
        cpp_int q = generateRandomPrimeNumberWithBits(bits);
        //欧拉函数值
        cpp_int phi = (p - 1) * (q - 1);
        //排除掉phi不和65537互素的情况（因为通常选取65537这个质数做密钥/公钥）
        if ((phi % 65537) == 0) {
            continue;
        }
        return generateKeyPair(p, q);
    }
}

std::pair<MyRsaKey, MyRsaKey>
MyRsaKey::generateKeyPair(boost::multiprecision::cpp_int p,

 boost::multiprecision::cpp_int q) {
    cpp_int n = p * q;
    cpp_int phi = (p - 1) * (q - 1);

    cpp_int e, d;

    cpp_int x, y;
    e = 65537;
    //使用拓展欧几里得算法求解模反元素特解
    auto gcd = extendedEuclidean(e, phi, x, y);

    //找出一个确保为正的模反元素
    if (x < 0) {
        d = x + ((-x) / (phi / gcd) + 1) * (phi / gcd);
    } else {
        d = x;
    }

    MyRsaKey k1(e, n);
    MyRsaKey k2(d, n);
    return {k1, k2};
}

cpp_int MyRsaKey::encode(cpp_int content){
    return powermod(content,e,n);
}
cpp_int MyRsaKey::decode(cpp_int content){
    return powermod(content,e,n);
}
```

main.cpp

```
#include <iostream>

#include "MyRsaKey.h"
using namespace std;
using boost::multiprecision::cpp_int;
int main() {
    cpp_int p, q, phi;
        while (1) {
            p = generateRandomPrimeNumberWithBits(1024);
            q = generateRandomPrimeNumberWithBits(1024);
```

```
            phi = (p - 1) * (q - 1);
            if ((phi % 65537) != 0) {
                break;
            }
        }
    cout<<"p="<<p<<endl;
    cout<<"q="<<q<<endl;
    cout<<"n="<<p*q<<endl;
    cout<<"phi(n)="<<phi<<endl;

    auto tmp = MyRsaKey::generateKeyPair(p,q);
    MyRsaKey key1 = tmp.first;
    MyRsaKey key2 = tmp.second;
    cout<<"key1={"<<key1.getE()<<","<<key1.getN()<<"}"<<endl;
    cout<<"key2={"<<key2.getE()<<","<<key2.getN()<<"}"<<endl;


    auto raw=generateRandomBigNumberWithBits(1024);
    cout<<"raw content: "<<raw<<endl;
    auto encoded=key1.encode(raw);
    cout<<"encrypted content "<<encoded<<endl;
    auto decoded=key2.decode(encoded);
    cout<<"decrypted content "<<decoded<<endl;

}
```