《第十一章 公钥密码数学基础》示例代码

作者：韩露露、杨波

日期：2019年3月1日

说明

本电子文档来源于书籍《深入浅出CryptoPP密码学库》，它最初被存放于GitHub上。任何人都可以复制、传播、使用本示例代码。

⇓

简介

《深入浅出CryptoPP密码学库》内容简介：

本书向读者介绍密码学库CryptoPP（或Crypto++）的使用方法和设计原理。CryptoPP是一个用C++语言编写的、开源的、免费的密码程序库，它最初由Wei Dai开发，现由开源社区维护。CryptoPP库广泛应用于学术界、开源项目、非商业项目以及商业项目，它几乎包括了目前已经公开的所有密码算法，支持当前主流的多种系统平台，并且具有良好的设计结构和较高的执行效率。

全书共15章，主要内容包括随机数发生器、Hash函数、流密码、分组密码、消息认证码、密钥派生和基于口令的密码、公钥加密系统、数字签名、密钥协商等，本书涵盖C++程序设计、设计模式、数论和密码学等知识。

本书最大的特点就是以应用为导向、以解决实际工程问题为目标，理论结合实践，将抽象的密码学变成保障信息安全的实际工具。

本书可以作为密码学、网络安全等专业在校学生的上机实验教材，也可以作为信息安全产品开发者、科研人员、密码算法实现者的参考手册。

⇓

资源

本书更多示例代码：https://github.com/locomotive-crypto
Crypto++网站：https://www.cryptopp.com/
Crypto++库GitHub地址：https://github.com/weidai11/cryptopp
Crypto++库SourceForge地址：https://sourceforge.net/projects/cryptopp/
Crypto++库Google论坛：
⇒公告通知地址：https://groups.google.com/forum/#!forum/cryptopp-announce
⇒用户群组地址：https://groups.google.com/forum/#!forum/cryptopp-users

目录

1 C/C++系统预定义的整数范围

下面的示例程序演示了C/C++中预定义的一些数据类型能够表示的范围。

```
#include <iostream>//使用cout、cin
#include <climits>//使用UCHAR_MAX、USHRT_MAX等
//使用numeric_limits<unsigned char>::max()、numeric_limits<unsigned short>::max()等
#include <limits>
using namespace std;//使用C++的命名空间std
int main()
{
    //C++,numeric_limits<unsigned char>::max();
    cout << "unsigned char 最大值: " << UCHAR_MAX <<
        ", 所占字节: " << sizeof(unsigned char) << endl;
    //C++,numeric_limits<unsigned short>::max();
    cout << "unsigned short 最大值: " << USHRT_MAX <<
        ", 所占字节: " << sizeof(unsigned short) << endl;
    //C++,numeric_limits<unsigned int>::max();
    cout << "unsigned int 最大值: " << UINT_MAX <<
        ", 所占字节: " << sizeof(unsigned) << endl;
    //C++,numeric_limits<unsigned long>::max();
    cout << "unsigned long 最大值: " << ULONG_MAX <<
        ", 所占字节: " << sizeof(unsigned long) << endl;
    //C++,numeric_limits<_ULONGLONG>::max();
    cout << "unsigned long long 最大值: " << ULLONG_MAX <<
        ", 所占字节: " << sizeof(unsigned long long) << endl;
    return 0;
}
```

执行程序，程序的输出结果如下：

```
unsigned char 最大值: 255, 所占字节: 1
unsigned short 最大值: 65535, 所占字节: 2
unsigned int 最大值: 4294967295, 所占字节: 4
unsigned long 最大值: 4294967295, 所占字节: 4
unsigned long long 最大值: 18446744073709551615, 所占字节: 8
请按任意键继续. . .
```

2 使用PrimeAndGenerator类

下面给出使用PrimeAndGenerator类产生特殊形式素数的示例：

```
#include<integer.h>//使用Integer
#include<iostream>//使用cout、cin
#include<osrng.h>//使用AutoSeededRandomPool
#include<nbtheory.h>//使用PrimeAndGenerator、VerifyPrime
using namespace std;//std是C++的命名空间
using namespace CryptoPP;//CryptoPP是CryptoPP库的命名空间
int main()
{
    AutoSeededRandomPool rng; //定义随机数发生器对象
    //定义PrimeAndGenerator对象，利用随机数发生器rng产生素数p和q
    //要求产生的p是1024比特的素数，q是512比特的素数
    PrimeAndGenerator pag(1,rng,1024,512);
    Integer p = pag.Prime(); //获取素数p的值
    Integer q = pag.SubPrime(); //获取素数q的值
    Integer r = (p-1)/q/2; //计算r的值，因为p=2*r*q+1，delta=1
    //打印p的值及比特数
    cout << "p(" << p.BitCount() << "): " << p << endl;
    //打印q的值及比特数
    cout << "q(" << q.BitCount() << "): " << q << endl;
    //打印r的值及比特数
    cout << "r(" << r.BitCount() << "): " << r << endl;
    if(VerifyPrime(rng,r,10)) //验证r是否为素数
    {
        //如果r为素数，则输出该信息
        cout << "r是素数" << endl;
    }
    else
    {
        //如果r不为素数，则输出该信息
        cout << "r不是素数" << endl;
    }
    return 0;
}
```

执行程序，程序的输出结果如下：

```
p(1024):
151041992509730008128659546823204358866799009601687789522858231971910939796114780291709181243854840999645258259075227757320549383746819820865261415047633942763158647760015899091227865743071351324802841234493390291424806337421453036386809816199757283698958980989066862482742706484815542870095265132985959423571.
q(512):
719007508949816563761780813238579077519956322424944747644701127986859289206997641821
```

```
22203678026508983788346566662284374041405256127211299502226520156560243.
g(1023):
65220114339011955689561415866240294395498234708554090332715574205084329615054620799
47317250647206471123106587547320746505806809137184262513911155452476754896103804059
34664937155383567132497071695746984067636401396612887220466497438439378384594203579
13019582522459124399913739997742719410637493914480461059428.
r(512):
10503505918202590601813666065118282958032225566470149717317702095471008591620502171
01102128607967771675180950529063239752415500262520054999939495670289099 5.
r不是素数
请按任意键继续. . .
```

3 算法综合使用示例及习题

3.1 构造BBS随机数发生器

BBS（Blum-Blum-Shub）发生器是已经证明过的密码强度最高的伪随机数发生器，它的示例代码如下：

```
#include<iostream> //使用cout、cin
#include<nbtheory.h> //使用数论等相关算法
#include<integer.h> //使用Integer
#include<osrng.h> //使用AutoSeededX917RNG
#include<aes.h> //使用AES算法-构造随机数发生器
using namespace std; //std是C++的命名空间
using namespace CryptoPP; //CryptoPP是CryptoPP库的命名空间
int main()
{
    //定义一个随机数发生器，用于产生随机的大整数
    AutoSeededX917RNG<AES> rng;
    Integer p,q,n; //定义3个大整数对象
    //产生512比特的大素数p和q，并且要求p=q=3 mod 4
    while(true)
    {
        //利用随机数发生器产生512比特的随机数p
        p.Randomize(rng,512);
        //利用概率型算法检测大整数p是满足要求
        if(VerifyPrime(rng,p,10) && (p % 4 == 3))
            break; //如果p为素数，且满足模4余3，则结束循环
    }
    while(true) //与选择参数p的原理一样
    {
        q.Randomize(rng,512);
        if(VerifyPrime(rng,q,10) && (q % 4 == 3))
            break;
    }
    n=p*q; //计算模数n的值
    Integer s,X;//BBS随机数发生器的种子
    while(true)
    {
        s.Randomize(rng,512); //产生一个512比特的随机种子
        if(RelativelyPrime(s,n))
            break; //如果s与n互素，满足要求，则结束循环
    }
    X = ModularExponentiation(s,2,n); //计算X的初值
    cout << "p=" << p << endl; //打印输出BBS发生器的第一个素数
    cout << "q=" << q << endl; //打印输出BBS发生器的第二个素数
    cout << "n=" << n << endl; //打印输出BBS发生器的模数
    cout << "s=" << s << endl; //打印输出BBS发生器的种子
```

```
41      cout << "产生200个随机比特: " ;
42      for( int i=0; i < 200 ; ++i)
43      {
44          X = ModularExponentiation(X,2,n);//迭代X的值
45          if( X % 2 == 1) //判断X的最后一比特的值
46              cout << "1 " ;
47          else
48              cout << "0 " ;
49      }
50      cout << endl ;
51      return 0;
52  }
```

执行程序，程序的输出结果如下：

```
p=11163396331512878849843078392905808230107133359595223065753183811314121791210809
51268996180591792572389433544312348473329625891891609586797588034468440379 1.
q=30921933944237520583420805181756170320318351218619302311975235651321307414902473
85615558075024071757702270805935824571094441543245162324726935457506631007.
n=34519380395638470178974098010976861971178210935830189489370089029527527129311931
42134206332557700821547864220604190471212242378379747493664363557176479842150661050
80252210883847576739578999930041615310603309403751277246457382987206568945252432741
64863377713681503030087784171890234397604185322084076828947537.
s=33842538549226591669035937226897950974247128895807837131078734864129770273404817
68926508644901930018708912459405810547887588243555794027014922001089091064.
产生200个随机比特: 1 1 1 0 0 1 1 1 1 0 1 0 0 0 1 1 1 0 1 0 1 1 1 0 1 1 0 0 0 1 1 0 0 1 1 0 0 1
1 0 0 0 1 0 0 1 1 0 0 0 0 0 1 1 1 1 1 0 1 1 1 1 0 0 1 0 0 1 1 0 1 1 0 0 0 1 0 0 1 1 1 0 0 1 1 0 0 1
1 1 1 1 1 0 1 1 1 0 0 1 0 1 0 0 0 0 0 0 0 0 0 1 1 1 1 0 1 0 0 1 0 0 1 0 1 0 0 0 0 0 1 1 0 1 1 1 0 0
1 1 1 1 1 1 0 0 1 0 0 1 1 1 1 0 1 0 0 0 1 0 1 0 1 0 1 0 1 1 1 1 0 0 1 1 1 1 0 1 1 1 0 1 1 1 0 0 0 0
1 0 0 0 1 1 0 1 1 1 1 0
请按任意键继续. . .
```

3.2 构造Rabin随机数发生器

下面是Rabin随机数发生器示例程序。

```
#include<iostream>//使用cout、cin
#include<integer.h>//使用Integer
#include<secblock.h>//使用SecBlockByte
#include<osrng.h>//使用AutoSeededRandomPool
#include<nbtheory.h>//使用VerifyPrime()等数论函数
using namespace std;//使用C++标准命名空间std
using namespace CryptoPP;//使用CryptoPP库的命名空间
//功能：产生一个指定长度的大素数
//参数prime：取回产生的素数
//参数length：要求产生的素数的长度
//返回值：无
void GeneratePrime(Integer& prime, size_t length);
int main()
{
    Integer p, q, n;//定义三个大整数对象
    GeneratePrime(p, 512);//产生512比特的随机素数p
    GeneratePrime(q, 512);//产生512比特的随机素数q
    n = p*q;//计算p和q的乘积
    const Integer half_n = n / 2;
    cout << "p: " << p << endl;//打印的值p
    cout << "q: " << q << endl;//打印的值q
    cout << "n: " << n << endl;//打印的值n
    //产生200各随机比特
    Integer x;//存储随机数发生器迭代过程的中间值
    AutoSeededRandomPool rng;//定义一个随机数发生器对象
    x.Randomize(rng, 512);//产生随机数发生器的初始种子
    cout << "产生200个随机数: " << endl;
    for (size_t i = 0; i < 200; ++i)
    {
        x *= x;//计算x的平方
        x %= n;//计算x模n的值
        if (x < half_n)
        {
            cout << (x % 2) << " ";//产生随机数
        }
        else
        {
            x = n -x;
            cout << ((n-x) % 2) << " ";//产生随机数
        }
    }
    cout << endl;
    return 0;
```

```
44 }
45 void GeneratePrime(Integer& prime, size_t length)
46 {
47     AutoSeededRandomPool rng;//定义一个随机数发生器对象
48     while (true)
49     {
50         prime.Randomize(rng, length);//产生一个指定长度的随机数
51         //验证产生的大整数是否为满足条件的大素数
52         if (VerifyPrime(rng, prime, 10) && (prime % 4 == 3))
53         {
54             break;
55         }
56     }
57 }
```

执行程序，程序的输出结果如下：

```
p: 68226499335276497246663647636834242314855303611621650586412409514805218978855405
64467071598835618526742506341488163737525124987939819849589221362110119 23.
q: 10715338148481710001721221736273230468686628834729174734812189796563585934891642
98176506446314756321704434438861231292258023799976238174955888829668813 3911.
n: 73107001106465028018722092438219936935331147653712306488993009985460495887621448
29064066803771850184803638364628108292576100415393199793027269419230962340540881168
86956382820181372489140637207840777490672506912446793040462786173552882991791508079
02014003640103004891515758259048091935719410165115638416208 53.
产生200个随机数:
1 0 0 0 1 0 0 0 0 1 0 1 0 1 1 1 1 1 1 0 1 1 0 0 0 0 1 0 0 0 0 1 0 1 1 1 1 1 1 0 1 1 0 1 1 1 1 0 0 1
1 0 1 1 1 0 1 1 0 1 0 0 0 0 0 1 0 1 1 1 0 1 1 1 1 0 1 1 1 1 0 0 0 1 0 0 0 0 1 1 0 0 1 0 0 0 0 0 0 1 0
0 0 1 1 1 0 1 1 1 1 1 1 0 0 1 0 0 0 0 1 1 0 1 1 1 1 0 0 0 1 1 1 1 1 0 0 0 0 0 1 1 0 1 0 0 1 0 0 0 1
0 0 1 0 0 0 1 1 1 0 0 1 0 0 0 0 1 1 1 0 0 1 1 0 0 1 0 0 0 0 0 1 0 1 1 1 0 0 0 1 0 1 0 1 1 0 1 0 1 0
请按任意键继续. . .
```

4 使用代数结构ECP类

题目要求如下：

设$P=(3,10)$，$Q=(9,7)$均为椭圆曲线$E_{23}(1,1)$上的两点，利用编程计算$P+Q$、$10P$、P的逆元以及椭圆曲线$E_{23}(1,1)$的单位元。

完整示例代码如下所示：

```
#include<iostream>//使用cout、cin
#include<ecp.h>//使用ECP
using namespace std;//使用C++标准命名空间std
using namespace CryptoPP;//使用CryptoPP库的命名空间
int main()
{
    ECP ecp(23, 1, 1); //定义一个ECP对象
    ECP::Point P(3, 10); //定义ecp(23,1,1)对象上的点P
    ECP::Point Q(9, 7); //定义ecp(23,1,1)对象上的点Q
    ECP::Point pResult = ecp.Add(P, Q); //获得点P和点Q相加的结果
    cout << "Add(P,Q).x = " << pResult.x << endl;
    cout << "Add(P,Q).y = " << pResult.y << endl;
    pResult = ecp.ScalarMultiply(P, Integer(10)); //计算10*P的结果
    cout << "ScalarMultiply(P, Integer(10).x = " << pResult.x <<
        endl;
    cout << "ScalarMultiply(P, Integer(10).y = " << pResult.y <<
        endl;
    ECP::Point identify = ecp.Identity(); //获得ecp(23,1,1)的单位元
    cout << "identify.x = " << identify.x << endl;
    cout << "identify.y = " << identify.y << endl;
    ECP::Point inverse = ecp.Inverse(P); //计算点P的逆元
    cout << "Inverse(P).x = " << inverse.x << endl;
    cout << "Inverse(P).y = " << inverse.y << endl;
    return 0;
}
```

执行程序，程序的输出结果如下：

```
Add(P,Q).x = 17.
Add(P,Q).y = 20.
ScalarMultiply(P, Integer(10).x = 6.
ScalarMultiply(P, Integer(10).y = 4.
identify.x = 0.
identify.y = 0.
Inverse(P).x = 3.
Inverse(P).y = 13.
请按任意键继续. . .
```

5 使用代数结构PolynomialMod2类

题目要求：
利用域的同构性质，尝试构造集合Z_8上的域，并用编程打印集合中元素的乘法运算表。
完整示例代码如下所示：

```
#include<iostream>//使用cout、cin
#include<gf2n.h>//使用PolynomialMod2
#include<iomanip>//使用setw(4)、setiosflags(ios::right)
#include<integer.h>//使用Integer
#include<secblock.h>//使用SecBlockByte
using namespace std;//使用C++标准命名空间std
using namespace CryptoPP;//使用CryptoPP库的命名空间
//功能：打印多项式及其对应的长整型数
//i_ploy：待转换的多项式
//返回值：无
void ConvertPolyToLong(const PolynomialMod2& i_poly)
{
    //定义SecByteBlock对象
    SecByteBlock buffer(i_poly.ByteCount());
    Integer n_decimal; //定义Integer对象
    //将多项式表示的字节型数据存储于buffer
    i_poly.Encode(buffer, buffer.size());
    //将buffer中的字节型数据解析成大整数
    n_decimal.Decode(buffer, buffer.size());
    //将大整数转换成long型数据
    long n = n_decimal.ConvertToLong();
    //打印多项式及其对应的整数
    cout << setw(4) << setiosflags(ios::right) << i_poly
        << "(" << n << ")" << " ";
}
int main()
{
    //表示多项式x^3+x+1
    const PolynomialMod2 tri_poly_mod(11, 4);
    //打印多项式
    cout << "tri_poly_mod=" << tri_poly_mod << endl;
    cout << "打印多项式模x^3+x+1，系数模2的乘法运算表" << endl;
    for (int i = 0; i < 8; ++i)
    {//依次打印0→7对应的多项式（第一行）
        PolynomialMod2 i_poly(i, 3);
        //打印多项式i_poly及其对应的整数
        ConvertPolyToLong(i_poly);
    }
    cout << endl; //换行
    for (int i = 0;i < 8; ++i)
    {
```

```
42          //下面两行代码实现依次打印0→7对应的多项式（第一列）
43          PolynomialMod2 i_poly(i,3); //定义多项式
44          //打印多项式i_poly及其对应的整数
45          ConvertPolyToLong(i_poly);
46          for (int j = 0; j < 8; ++j)
47          {//打印多项式模x^3+x+1，系数模2的乘法表
48              PolynomialMod2 j_poly(j,3); //定义多项式
49              PolynomialMod2 mod_result = (i_poly * j_poly).Modulo(
                    tri_poly_mod);
50              //打印多项式mod_result及其对应的整数
51              ConvertPolyToLong(mod_result);
52          }
53          i_poly += PolynomialMod2::One(); //多项式累加1
54          cout << endl; //换行
55      }
56      cout << "打印多项式模x^3+x+1，系数模2的元素的逆元" << endl;
57      for (int i = 1; i < 8; ++i)
58      {
59          PolynomialMod2 i_poly(i, 3); //定义多项式
60          //打印多项式i_poly及其对应的整数
61          ConvertPolyToLong(i_poly);
62          //计算乘法逆元
63          PolynomialMod2 i_poly_inver = i_poly.InverseMod(tri_poly_mod
                );
64          //打印多项式i_poly_inver及其对应的整数
65          ConvertPolyToLong(i_poly_inver);
66          cout << endl; //换行
67      }
68      return 0;
69  }
```

执行程序，程序的输出结果如下：

```
tri_poly_mod=1011b
打印多项式模x^3+x+1，系数模2的乘法运算表
          0b(0)  1b(1)   10b(2)   11b(3)  100b(4)  101b(5)  110b(6)  111b(7)
0b(0)     0b(0)  0b(0)   0b(0)    0b(0)   0b(0)    0b(0)    0b(0)    0b(0)
1b(1)     0b(0)  1b(1)   10b(2)   11b(3)  100b(4)  101b(5)  110b(6)  111b(7)
10b(2)    0b(0)  10b(2)   100b(4)  110b(6) 11b(3)   1b(1)    111b(7)  101b(5)
11b(3)    0b(0)  11b(3)   110b(6)  101b(5) 111b(7)  00b(4)   1b(1)    10b(2)
100b(4)   0b(0)  100b(4)  11b(3)   111b(7) 110b(6   10b(2)   101b(5)  1b(1)
101b(5)   0b(0)  101b(5)  1b(1)    100b(4) 10b(2)   111b(7)  11b(3)   110b(6)
110b(6)   0b(0)  110b(6)  111b(7)  1b(1)   101b(5)  11b(3)   10b(2)   100b(4)
111b(7)   0b(0)  111b(7)  101b(5)  10b(2)  1b(1)    110b(6)  100b(4)  11b(3)
打印多项式模x^3+x+1，系数模2的元素的逆元
1b(1)    1b(1)
```

```
10b(2)   101b(5)
11b(3)   110b(6)
100b(4)  111b(7)
101b(5)  10b(2)
110b(6)  11b(3)
111b(7)  100b(4)
请按任意键继续. . .
```

6 声明

Cryptography

⇓

⇓

⇓

此为《深入浅出CryptoPP密码学库》随书电子文档，它仅包含书籍中示例程序的源代码。关于示例代码的解释说明，详见书籍相应章节内容。

由于作者水平有限，错误之处在所难免。欢迎通过如下方式反馈相关问题：

⇒ QQ：1220195669

⇒ 微信：cc1220195669

⇓

⇓

⇓

《深入浅出CryptoPP密码学库》