《第四章 初识CryptoPP库》示例代码

作者：韩露露、杨波

日期：2019年9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 使用帮助文档

使用#include指令把大整数类所在的头文件integer.h包含进源代码文件，之后就可以像使用预定义的数据类型那样使用Integer类。

```
#include<iostream> //使用cout、cin
#include<integer.h> //使用Integer
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()
{
    Integer big_number("12345678909876543 21");
    cout << "big_number=" << big_number << endl;
    return 0;
}
```

执行程序，程序的输出结果如下：

```
big_number=1234567890987654321.
请按任意键继续. . .
```

2 向CryptoPP库添加随机数发生器算法

下面演示如何向CryptoPP库添加随机数发生器算法–MyRNG类，它的描述如下：

MyRNG类是一个随机数发生器算法，它根据外部输入的种子（两个数值）来产生随机序列。用f1、f2分别表示MyRNG的内部状态，用a、b分别表示外部输入的种子，则产生随机序列的过程可描述为：

初始化内部状态：

f1=a;

f2=b;

产生随机序列s1、s2、…的过程：

s1=f1+f2;

f1=f2;

f2=s1;

s2=f1+f2;

f1=f2;

f2=s2;

…

首先，MyRNG类应该继承自RandomNumberGenerator类，用Integer类表示它的内部状态，通过构造函数获得外部种子并初始化它的内部状态。

```
class MyRNG :public RandomNumberGenerator
{
public:
    explicit MyRNG(const Integer& a, const Integer& b)
    {//利用外部种子初始化RNG内部状态
        f1 = a;
        f2 = b;
    }
private:
    Integer f1, f2; //内部状态
};
```

然后，依次重写基类中的AlgorithmName()函数、GenerateBlock()函数、CanIncorporateEntropy()函数、IncorporateEntropy()函数。

（1）重写AlgorithmName()函数——Algorithm类有一个虚函数AlgorithmName()，用于返回算法的标准名字。在默认情况下，该函数返回字符串"unknown"。

```
std::string AlgorithmName() const
{
    return "MyRNG Created by Lulu";// 返回算法标准的名字
}
```

（2）重写GenerateBlock()函数

```
void GenerateBlock(byte *output, size_t size)
{//产生所需长度的随机序列
    Integer s; //存储产生的序列
```

```
    size_t curlen;
    while (size > 0)
    {
        s = f1 + f2; //产随机序列
        f1 = f2; //更新内部状态f1
        f2 = s; //更新内部状态f2
        curlen = s.ByteCount(); //s被编码成字节数据后所占的长度
        //计算还需向output指向的缓冲区存放的字节数
        curlen = curlen < size ? curlen : size;
        s.Encode(output, curlen); //从s中取回curlen字节长度的数据
        size -= curlen; //计算还需取回的字节数
        output += curlen; //让output指向下一段待取回数据的缓冲区
    }
}
```

若MyRNG算法允许接受外部输入的熵，则还需要重写CanIncorporateEntropy()和IncorporateEntropy()。否则，我们什么也不用做。现在，我们假定MyRNG允许有外部的熵输入。对于前者，仅需要将函数的返回值设置为true即可。对于后者，需要考虑外部输入熵如何影响MyRNG的内部状态。外部输入熵以字符串（长度为size字节）的形式传递给MyRNG，在这里我们对它做如下处理：

将输入熵前size/2字节长的字符串解码为大整数a，将输入熵中剩余部分的字符串解码为大整数b，分别将大整数a和b累加至f1和f2（简单模拟用外部输入熵更新随机数发生器的内部状态）。

（3）重写CanIncorporateEntropy()和IncorporateEntropy()函数

```
bool CanIncorporateEntropy() const
{
    return true; //允许该随机数发生器有外部熵输入
}
void IncorporateEntropy(const byte *input, size_t length)
{
    //以下操作实现将外部输入的前后两部分解码成大整数
    size_t lenhalt = length / 2;
    Integer a, b;
    a.Decode(input, lenhalt); //将前size/2字节长的字符串解码为大整数a
    //将剩余部分的字符串解码为大整数b
    b.Decode(input + lenhalt, length - lenhalt);
    //利用外部输入更新该RNG的内部状态
    f1 += a; //更新内部状态f1
    f2 += b; //更新内部状态f2
}
```

最终，MyRNG类的完整定义如下：

```
class MyRNG :public RandomNumberGenerator
{
public:
    explicit MyRNG(const Integer& a, const Integer& b)
```

```
5       {
6           f1 = a;
7           f2 = b;
8       }
9       virtual void GenerateBlock(byte *output, size_t size)
10      {
11          Integer s;
12          size_t curlen;
13          while (size > 0)
14          {
15              s = f1 + f2;
16              f1 = f2;
17              f2 = s;
18              curlen = s.ByteCount();
19              curlen = curlen < size ? curlen : size;
20              s.Encode(output, curlen);
21              size -= curlen;
22              output += curlen;
23          }
24      }
25      virtual std::string AlgorithmName() const
26      {
27          return "MyRNG Created by Lulu";
28      }
29      virtual bool CanIncorporateEntropy() const
30      {
31          return true;
32      }
33      virtual void IncorporateEntropy(const byte *input, size_t length
            )
34      {
35          size_t lenhalt = length / 2;
36          Integer a, b;
37          a.Decode(input, lenhalt);
38          b.Decode(input + lenhalt, length - lenhalt);
39          f1 += a;
40          f2 += b;
41      }
42  private:
43      Integer f1, f2;
44  };
```

MyRNG类的使用方式与CryptoPP库中已有的随机数发生器算法使用方法一样，详见第五章。

3 CryptoPP库的Base系列编码算法

下面使用CryptoPP库的Base系列编码算法来编码字符串"I like Cryptography."。

```
#include<iostream> //使用cout、cin
#include<string> //使用string
#include<files.h> //使用FileSink
#include<hex.h> //使用HexEncoder
#include<base32.h> //使用Base32Encoder
#include<base64.h> //使用Base64Encoder
#include<filters.h> //使用StringSource
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()
{
    string message = "I like Cryptography."; //待编码的字符串
    cout << "Base64: "; //打印字符串的Base64编码
    StringSource Base64Src(message, true, new Base64Encoder(new
        FileSink(cout)));
    cout << endl << "Base64url: "; //打印字符串的Base64url编码
    StringSource Base64rulSrc(message, true, new Base64URLEncoder(
        new FileSink(cout)));
    cout << endl << "Base32: "; //打印字符串的Base32编码
    StringSource Base32Src(message, true, new Base32Encoder(new
        FileSink(cout)));
    cout << endl << "Base32hex: "; //打印字符串的Base32hex编码
    StringSource Base32hexSrc(message, true, new Base32HexEncoder(
        new FileSink(cout)));
    //打印字符串的Base16编码，也即数据的十六进制编码
    cout << endl << "Base16: ";
    StringSource Base16Src(message, true, new HexEncoder(new
        FileSink(cout)));
    cout << endl;
    return 0;
}
```

执行程序，程序的输出结果如下：

```
Base64：SSBsaWtlIENyeXB0b2dyYXBoeS4=
Base64url：SSBsaWtlIENyeXB0b2dyYXBoeS4
Base32：JESG24MMNWSEG6V3QB4G835UNF2GS8JQ
Base32hex：94G6OQBBCKG46SJPE1Q6UPRIC5O6GU9E
Base16：49206C696B652043727970746F6772617068792E
请按任意键继续. . .
```

关于本程序的详细说明，详见本章Pipeling范式数据处理部分相关内容。

4 CryptoPP库的ASN.1系列编码算法

下面演示如何使用DER编码大整数和公钥密码系统的密钥。

```
#include<iostream> //使用cout、cin
#include<integer.h> //使用Integer
#include<files.h> //使用FileSource
#include<osrng.h> //使用AutoSeededRandomPool
#include<rsa.h> //使用RSA
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()
{
   Integer BigNumer; //定义一个大整数对象
   AutoSeededRandomPool rng; //定义一个随机数发生器
   BigNumer.Randomize(rng,512); //利用随机数发生器，产生随机的512比特整数
   //将产生的大整数以DER编码方式编码，并存储于BigNumer.der文件中
   BigNumer.DEREncode(FileSink("BigNumber.der").Ref());
   RSA::PrivateKey privateKey; //定义一个RSA私钥对象
   privateKey.Initialize(rng,1024); //产生一个随机的模为1024比特的RSA私钥
   //将产生的私钥以DER编码方式编码，并存储于prikey.der文件中
   privateKey.DEREncode (FileSink("prikey.der").Ref());
   return 0;
}
```

本示例程序先定义了一个随机数发生器对象，接着用该随机数发生器分别产生一个512比特的大整数对象和一个模数为1024比特的RSA私钥对象。然后，使用DER编码规则对它们进行编码，并将编码的结果分别存储于BigNumber.der文件和prikey.der文件。

由于DER编码结果并非是有意义、可辨识的“文本”，所以需要使用特定的“阅读器”来解析编码结果。我们使用由Peter Gutmann[1]编写的ASN1“阅读器”工具来查看文件BigNumber.der和文件prikey.der的内容。

BigNumber.der文件的内容：

```
0   65: INTEGER
      :   00 CC 95 0E 05 13 87 0B 32 2D C0 AD 15 99 5C FF
      :   A3 07 27 97 87 4B 30 69 A5 01 62 BB C6 2D 06 D3
      :   BB 66 CE 59 1D 65 11 AC 48 AE 04 BB 9D CF 8B 61
      :   98 40 7E 4A F3 67 FD 6A 0A C2 25 41 DA C4 B8 B7
      :   A1
```

prikey.der文件的内容：

```
0 1182: SEQUENCE {
   4    1:   INTEGER 0
   7   13:   SEQUENCE {
   9    9:     OBJECT IDENTIFIER '1 2 840 113549 1 1 1'
  20    0:     NULL
```

[1]https://www.cs.auckland.ac.nz/~pgut001/dumpasn1.c

```
  :       }
 22 1160: OCTET STRING, encapsulates {
 26 1156:   SEQUENCE {
 30    1:     INTEGER 0
 33  252:     INTEGER
  :       37 07 ED 5B 23 66 AD 1A AA F7 36 22 48 21 93 71
  :       ED 68 EC 76 6A 96 9C 6F E6 2C 12 4F 08 AC 4A C2
  :       B1 5D EF B9 9D B8 57 BB 35 04 33 11 C9 89 5C 65
  :       A2 9D 6A AD DE 65 DF 39 00 C0 C7 1B 22 E5 29 BC
  :       15 06 38 08 07 60 58 8E 84 BB E2 31 D5 1B 62 34
  :       F8 E1 C1 94 AD F8 BA 45 54 38 21 EF 0E 2E 65 98
  :       3F F2 63 AE 17 10 EA A9 74 87 BC 24 58 23 50 DC
  :       DE DB 6B 9F 5F 53 7E 99 AD 97 8A EC D9 A2 FF 84
  :             [ Another 124 bytes skipped ]
288    1:     INTEGER 17
291  252:     INTEGER
  :       06 79 67 37 E6 0C 14 5D 7D 86 7E D6 DB 4F 3E 85
  :       DF B1 FD B3 94 11 B8 0D 2A 23 4D 72 B5 B9 EA AD
  :       7E 47 49 61 21 9D 37 7F 6F A6 24 20 35 D3 EC C0
  :       A9 B8 2A AB 0B 1B 0B 33 E1 F8 8F E5 13 2A 04 E8
  :       F3 6A 24 B5 A6 83 CE 2E E2 70 74 F6 CD C6 FC 7E
  :       B3 DE 53 02 6E D1 F7 CB EB CA 5E 58 5C 05 75 5D
  :       34 B3 1A C9 2F E3 DF 5F 3A E2 CA D7 19 6D 91 0A
  :       ED 0A C1 5E 0B 36 FF D5 D8 2F F2 39 FB 7C 7A 97
  :             [ Another 124 bytes skipped ]
546  126:     INTEGER
  :       73 43 74 BE 30 B8 25 FB 1E 00 D6 61 C0 5B C5 FE
  :       80 00 40 BE 6B 64 06 58 42 15 34 7D E2 D9 FB FC
  :       56 9C 5F 72 3B E5 D7 38 E1 89 AA F9 19 AD 23 FD
  :       57 82 B2 7E 6C 5A 19 0C BD 2E 53 E8 1D 9A 76 A6
  :       45 70 EB 38 3A 19 79 48 60 0B 84 88 C6 A8 F3 30
  :       96 1B 11 0C DE 6D B1 CB D1 C5 FA 1B C0 7E 4E 76
  :       9D BD C8 73 40 8C B1 D2 FA E3 7A A6 4D 46 95 AF
  :       80 53 4B B1 3D 0E DB 61 91 0F D4 AD 21 13
674  126:     INTEGER
  :       7A 39 41 F7 5F BF 3F 3F 49 93 2F E7 AF 38 F5 4A
  :       84 6B 59 4B 0E 8F 34 E4 D8 D1 2E F9 31 FD FC 47
  :       C7 0F B9 3D 22 3E 1E 40 86 BD 15 8C EE 6C C9 A3
  :       04 62 72 7B 26 C0 B6 9E 0A 4A 50 5A 0D 66 D9 E0
  :       2E 39 54 02 52 77 2A F1 33 E3 EB EA 09 AE B0 A0
  :       9A A8 8E 1E 89 37 68 E6 32 54 51 B0 70 C9 37 11
  :       5D 43 C6 E8 DE 1D 59 9A 3F 48 FE AA D3 A5 B1 83
  :       52 45 99 86 8B BF 9D 96 56 F7 26 9B E5 79
802  126:     INTEGER
  :       28 AE 65 70 4D 6E 2B 85 CE 5A A6 04 62 02 45 E1
  :       5A 5A 71 34 25 E7 11 4C 53 8F 03 77 B9 7A 1C B3
```

```
                :              69 DC D6 64 8D 9C 6A 14 13 5D C3 DF 72 79 57 FF
                :              0F D3 C6 86 F9 10 BD 8C 06 88 D2 51 EC 54 A2 58
                :              CD 36 E9 9B 5F CC C1 64 D6 9A A7 3F 55 2C 92 11
                :              25 EB 6F 6D F4 26 B7 38 E0 A0 3A 27 E9 95 FD 93
                :              46 BB 73 EC 71 22 99 1D 49 7D 76 95 0C 37 07 A7
                :              5A 77 C0 5C AC 23 5C 7C C9 C9 5A 1E FC 9D
 930    126:           INTEGER
                :              4F 15 FD 81 F2 A8 EC B0 7A E6 C4 A4 F8 E8 9E B7
                :              BF 18 48 D6 36 98 E5 FD 7D 3C 0F 55 F3 2B DF 79
                :              BD 0A 2C 90 F8 0A 13 93 2A 01 E0 C4 9A 46 64 5A
                :              6C 3F B3 7C DC D7 0C C0 9D 3F 24 EE F9 9C E7 54
                :              D2 9D 90 B6 35 5C 2A D8 4E C0 A7 B5 8D CB 63 3A
                :              BE 6D 10 AA 58 C9 80 1C 7A EB 43 EA A3 55 05 83
                :              B4 D1 80 B4 CB F4 DF A0 0A D4 E1 05 1F 89 54 BE
                :              62 69 45 38 F1 03 84 15 FC 09 55 37 B2 99
1058    126:           INTEGER
                :              51 F5 2E D7 31 53 94 5D CD 58 C3 F4 34 54 BA CC
                :              FC 2C 13 FE B9 87 D3 F0 AC 10 9F 03 F2 9F 1C 7B
                :              77 4C D3 32 08 6C 54 CF A9 47 A9 F5 9A 57 EF 69
                :              4F 82 DD 0F 13 C4 C2 FA CB E2 A6 15 40 0A A2 28
                :              D0 2C AE 23 C9 2E 24 D4 B2 94 00 F2 49 BF B5 C7
                :              20 D1 6B 14 D5 3F 7E E8 72 E0 51 A8 B9 AD 43 11
                :              A5 ED EF C1 CB 0C 75 61 6E 0A 4A B1 B9 FE EA D1
                :              B5 BE 75 A9 FF CD E2 64 5C 2F 91 9E 35 59
                :          }
                :      }
                :  }
```

5 Pipeling范式数据处理原理

5.1 Pipeling范式数据链中动态创建对象自动销毁的原理

下面的示例程序模拟了Pipeling范式数据链中动态创建对象自动销毁的原理。

```
#include<iostream> //使用cout、cin
#include<smartptr.h> //使用member_ptr
using namespace std; //std是C++的命名空间
using namespace CryptoPP; //CryptoPP是CryptoPP库的命名空间
class Buffer //模拟BufferedTransformation的作用
{
public:
    virtual ~Buffer(){} //虚的析构函数
};
class Source:public Buffer //模拟CryptoPP库中的Source类
{
public:
    Source(Buffer* buffer):m_buffer(buffer){}
    virtual ~Source()
    {
        cout << "Source类对象析构" << endl; //析构时打印输出提示信息
    }
protected:
    member_ptr<Buffer> m_buffer; //智能指针数据成员
};
class Filter:public Buffer //模拟CryptoPP库中的Filter类
{
public:
    Filter(Buffer* buffer):m_buffer(buffer){}
    virtual ~Filter()
    {
        cout << "Filter类对象析构" << endl; //析构时打印输出提示信息
    }
protected:
    member_ptr<Buffer> m_buffer; //智能指针数据成员
};
class Sink:public Buffer //模拟CryptoPP库中的Sink类
{
public:
    Sink(Buffer* buffer):m_buffer(buffer){}
    virtual ~Sink()
    {
        cout << "Sink类对象析构" << endl; //析构时打印输出提示信息
    }
protected:
    member_ptr<Buffer> m_buffer;//智能指针数据成员
```

```
};
int main()
{
    //测试
    {//进入作用域–Src对象构造
        //模拟CryptoPP库中Source、Filter和Sink的实现原理
        Source Src(new Filter( new Sink(nullptr)));
    }//离开作用域–Src对象析构
    getchar(); //暂停–等待输入
    return 0;
}
```

执行程序，程序的输出结果如下：

```
Source类对象析构
Filter类对象析构
Sink类对象析构
```

本示例模拟了CryptoPP库中Source、Filter和Sink类的实现原理。虽然程序的主函数中只有三行代码（确切地说，只有一行），但是这个程序深刻揭示了Pipeling范式数据处理中数据链是如何形成的，程序的输出结果解释了“为什么不需要用户显式地释放动态创建的对象”。

6 以自动方式使用Pipeling范式技术

6.1 以十六进制编码文件

下面的示例程序将文件test.txt的内容以十六进制形式编码，并把编码的结果存储于文件encoder.txt中。

```
1  #include<iostream> //使用cout、cin
2  #include<files.h> //使用FileSource、FileSink
3  #include<hex.h> //使用HexEncoder
4  using namespace std; //std是C++的命名空间
5  using namespace CryptoPP; //CryptoPP是CryptoPP库的命名空间
6  int main()
7  {
8      //将文件test.txt的内容以十六进制形式编码,
9      //并把编码的结果存储于文件encoder.txt中
10     FileSource fSrc("test.txt",true,new HexEncoder(new FileSink("
           encoder.txt")));
11     return 0;
12 }
```

当程序执行完毕后，文件encoder.txt中的内容即为文件test.txt内容对应的十六进制表示形式。

6.2 以十六进制编码字符串

下面的示例程序将字符串instr中的字符转换成十六进制后存入hexstr。

```
#include<iostream> //使用cout、cin
#include<filters.h> //使用StringSource、StringSink
#include<hex.h> //使用HexEncoder
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()
{
    string instr = "I like cryptography."; //定义一个string对象
    string hexstr;
    //将字符串instr中的字符转换成十六进制后存入hexstr
    StringSource strSrc(instr,true,new HexEncoder(new StringSink(
        hexstr)));
    cout << "instr=" << instr << endl; //打印输出
    cout << "hexstr=" << hexstr << endl; //打印输出
    return 0;
}
```

执行程序，程序的输出结果如下：

```
instr=I like cryptography.
hexstr=49206C696B652063727970746F6772617068792E
请按任意键继续. . .
```

7 以手动方式使用Pipeling范式技术

下面以手动方式使用StringSource、HexEncoder和FileSink类对象。

```
#include<iostream> //使用cout、cin
#include<files.h>> //使用FileSink
#include<hex.h> //使用HexEncoder
#include<string> //使用string
#include<secblock.h> //使用SecByteBlock
#include<filters.h> //使用StringSource
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()
{
    string message = "I like Cryptography."; //定义一个string对象
    //以手动方式使用Source类对象–以message对象为真实的Source
    StringSource msg_str_src(message, true); //定义一个StringSource对象
    SecByteBlock sec_get(4); //定义一个SecByteBlock对象
    cout << "StringSource对象中可取回的字符数: "
        << msg_str_src.MaxRetrievable() << endl;
    msg_str_src.Skip(2); //跳过Source中的前两个字符, 即"I "
    msg_str_src.Get(sec_get, sec_get.size());//从Source中取出4个字符
    cout << "StringSource对象中可取回的字符数: "
        << msg_str_src.MaxRetrievable() << endl;
    //以手动方式使用Filter类对象
    HexEncoder hex_enc_src; //定义一个Filter对象
    cout << "HexEncoder对象中可取回的字符数: "
        << hex_enc_src.MaxRetrievable() << endl;
    //将从Source取出的数据存入Filter
    hex_enc_src.Put(sec_get, sec_get.size());
    cout << "HexEncoder对象中可取回的字符数: "
        << hex_enc_src.MaxRetrievable()<< endl;
    //以手动方式使用Sink类对象–以cout对象为真实的Sink
    FileSink cout_sink(cout); //定义一个Sink对象
    //将从Source对象中取出的数据存入Sink对象中
    cout_sink.Put(sec_get, sec_get.size()); //打印输出sec_get中的内容
    sec_get.resize(hex_enc_src.MaxRetrievable()); //重置sec_get的大小
    hex_enc_src.Get(sec_get, sec_get.size()); //从Filter中取出8个字符
    cout << endl;
    //将从Filter对象中取出的数据存入Sink对象中
    cout_sink.Put(sec_get, sec_get.size()); //打印输出sec_get中的内容
    cout << endl;
    return 0;
}
```

执行程序，程序的输出结果如下：

```
StringSource对象中可取回的字符数：20
StringSource对象中可取回的字符数：14
HexEncoder对象中可取回的字符数：0
HexEncoder对象中可取回的字符数：8
like
6C696B65
请按任意键继续. . .
```

8 以半手动或半自动方式使用Pipeling范式技术

下面的示例程序演示了如何向数据链添加节点，如何以半手动或半自动方式使用Pipeling范式技术。

```
#include<iostream> //使用cout、cin
#include<hex.h> //使用HexEncoder
#include<files.h>> //使用FileSink
#include<string> //使用string
#include<filters.h> //使用StringSource
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()
{
    try
    {
        string message = "I like Cryptography."; //定义一个string对象
        StringSource strSrc(message, false); //定义一个StringSource对象
        if (strSrc.Attachable()) //判断该对象是否允许其他对象附着
            cout << "允许向StringSource附着新的数据链节点" << endl;
        else
            cout << "不允许向StringSource附着新的数据链节点" << endl;
        FileSink cout_sink(cout); //定义一个FileSink对象
        if (cout_sink.Attachable()) //判断该对象是否允许其他对象附着
            cout << "允许向FileSink附着新的数据链节点" << endl;
        else
            cout << "不允许向FileSink附着新的数据链节点" << endl;
        //向strSrc对象附着一个依次由HexEncoder和FileSink对象组成的数据链
        strSrc.Attach(new HexEncoder(new FileSink(cout)));
        strSrc.PumpAll(); //泵出strSrc中所有数据，即字符串message
        cout << endl;
        HexEncoder hexEnc; //定义一个HexEncoder对象
        //向hexEnc对象附着一个FileSink类关联的cout对象
        hexEnc.Attach(new FileSink(cout));
        //让字符串数据流message流经HexEncoder，并流至FileSink类对象
        hexEnc.Put(reinterpret_cast<CryptoPP::byte*>(&message[0]),
            message.size());
        cout << endl;
        string strnew;//定义一个string对象
        hexEnc.Detach(new StringSink(strnew)); //分离原节点，附着新节点
        //让字符串数据流message流经HexEncoder，并流至FileSink类关联的strnew对象
        hexEnc.Put(reinterpret_cast<CryptoPP::byte*>(&message[0]),
            message.size());
        cout << strnew << endl; //打印输出strnew对象中的内容
    }
    catch (const Exception& e)
    {//出现异常
```

```
41            cout << e.what() << endl; //异常原因
42        }
43        return 0;
44    }
```

执行程序，程序的输出结果如下：

```
允许向StringSource附着新的数据链节点
不允许向FileSink附着新的数据链节点
49206C696B652043727970746F6772617068792E
49206C696B652043727970746F6772617068792E
49206C696B652043727970746F6772617068792E
请按任意键继续. . .
```

9 单链型到多链型Pipeling范式数据处理

下面的示例演示了Redirector类和ChannelSwitch类的使用方法。本示例以一个string对象为数据源Source，利用重定向器让Source中流出的数据分叉至ChannelSwitch对象关联的三条数据链。它们的作用分别是，让Source中的数据直接输出至标准输出设备，将Source中的数据转换成Base32编码并存储到string对象中，将Source中的数据转换成Base64编码并存储到string对象中。

```
#include<iostream> //使用cout、cin
#include<filters.h> //使用StringSource、StringSink
#include<base32.h> //使用Base32Encoder
#include<base64.h> //使用Base64Encoder
#include<files.h> //使用FileSink
#include<channels.h> //使用ChannelSwitch
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()
{
    string message = "I like Cryptography.";
    string str1,str2; //定义两个string对象，用于存放编码后的字符串
    FileSink fsink(cout); //将标准输出设备当做Sink
    //将数据以Base32编码后存入str1对象中
    Base32Encoder b32enc(new StringSink(str1));
    //将数据以Base64编码后存入str2对象中
    Base64Encoder b64enc(new StringSink(str2));
    ChannelSwitch cs; //定义一个ChannelSwitch对象
    cs.AddDefaultRoute(fsink); //添加第一条数据链
    cs.AddDefaultRoute(b32enc); //添加第二条数据链
    cs.AddDefaultRoute(b64enc); //添加第三条数据链
    cout << "message="; //打印输出string对象message中的内容
    //将数据源中的数据同时泵出至多条数据链
    StringSource ss(message,true,new Redirector(cs));
    //打印输出string对象str1中的内容
    cout << endl << "str1=" << str1 << endl;
    cout << "str2=" << str2 << endl; //打印输出string对象str2中的内容
    return 0;
}
```

执行程序，程序的输出结果如下：

```
message=I like Cryptography.
str1=JESG24MMNWSEG6V3QB4G835UNF2GS8JQ
str2=SSBsaWtlIENyeXB0b2dyYXBoeS4=
请按任意键继续. . .
```

10 计时器工具

下面以Timer类为例演示计时器类算法的使用方法。

```
#include<iostream> //使用cout、cin
#include<hrtimer.h> //使用Timer
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Timer对象，并且设置计时器以秒为单位
    Timer tm(TimerBase::SECONDS);
    cout << "当前计数器的值: " << tm.GetCurrentTimerValue() << endl;
    cout << "当前计数器的值: " << tm.GetCurrentTimerValue() << endl;
    size_t sum=0;
    tm.StartTimer(); //开始计时
    for(size_t i=0; i < 0xffffffff-1;++i)
        sum+=i; //待统计执行时间的语句
    cout << "执行(0xffffffff-1)次加法花费的时间: "
        << tm.ElapsedTimeAsDouble() << endl;
    size_t mul=1;
    tm.StartTimer(); //开始计时
    for(size_t i=0; i < 0xffffffff-1;++i)
        mul*=i; //待统计执行时间的语句
    cout << "执行(0xffffffff-1)次乘法花费的时间: "
        << tm.ElapsedTimeAsDouble() << endl;
    return 0;
}
```

执行程序，程序的输出结果如下：

```
当前计数器的值: 60797074931
当前计数器的值: 60797730934
执行(0xffffffff-1)次加法花费的时间: 0.00952409
执行(0xffffffff-1)次乘法花费的时间: 4.46219e-007
请按任意键继续. . .
```

11 秘密分割工具

下面以Shamir秘密共享算法为例，首先演示将一个文件分割成多个文件的方法。然后，演示如何用分割后的这些文件中的一定份额恢复出原始文件。

```
#include<iostream> //使用cout、cin
#include<string> //使用string
#include<files.h> //使用FileSource
#include<osrng.h> //使用AutoSeededRandomPool
#include<ida.h> //使用SecretSharing
#include<channels.h> //使用ChannelSwitch
using namespace std; //std是C++的命名空间
using namespace CryptoPP; //CryptoPP是CryptoPP库的命名空间
//功能：将filename表示的文件分割成nShares个份额，
//  并且当有threshold个或更多的这些份额时，可以恢复filename文件的内容
//参数threshold：设置秘密分割时需要的陷门数量
//参数nShares：将文件filename分割的份额数
void SecretShareFile(int threshold,int nShares,const string
   inFilenames);
//功能：根据inFilenames数组表示的文件恢复出原始文件，
//  并且将恢复后的文件命名为outFilename
//参数threshold：输入陷门的个数，即inFilenames数组的长度
//参数outFilename：文件恢复后的名字
//参数inFilenames：string数组，表示输入的threshold文件名字
void SecretRecoverFile(int threshold,const string& outFilename,const
    vector<string>& inFilenames);
int main()
{
    try
    {
        string filename; //待分割文件的名字
        int threshold, nShares; //陷门数和份额数
        cout << "待分割文件的名字：" ;
        getline(cin,filename);
        cout << "陷门值（threshold）：";
        cin >> threshold;
        cout << "分割的份额数（nShares）：";
        cin >> nShares;
        SecretShareFile(threshold,nShares,filename); //执行文件分割
        cout << "恢复文件时，拥有的陷门数：" ;
        cin >> threshold;
        cin.sync(); //清空输入流缓存
        vector<string> inFilenames; //string数组，存储输入的文件名
     for(int i=0; i < threshold;++i)
        {
            cout << "输入文件名：" ;
            string file;
```

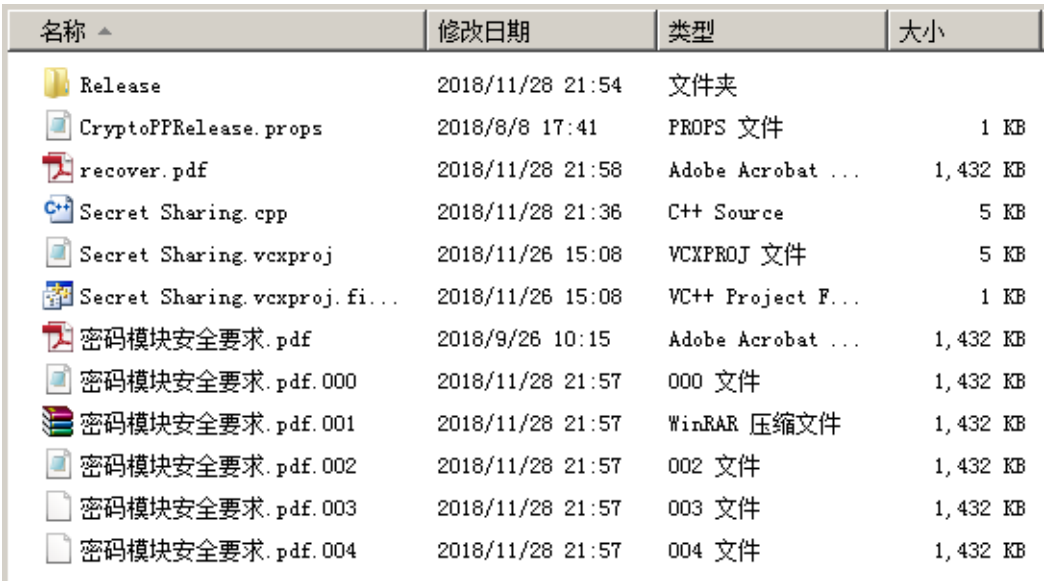
```
41                  getline(cin,file);
42                  inFilenames.push_back(file);
43              }
44              cout << "设置恢复后的文件名: " ;
45              getline(cin,filename);
46              //执行文件恢复
47              SecretRecoverFile(threshold,filename,inFilenames);
48      }
49      catch(const Exception& e)
50      {//出现异常
51          cout << e.what() << endl; //异常原因
52      }
53      return 0;
54  }
55  void SecretShareFile(int threshold,int nShares,const string filename
        )
56  {
57      //nShares的范围为[1,1000]
58      CRYPTOPP_ASSERT(nShares >= 1 && nShares<=1000);
59      if (nShares < 1 || nShares > 1000) //份额输入错误，则抛出异常
60          throw InvalidArgument("SecretShareFile: "
61              +IntToString(nShares)+" is not in range [1, 1000]");
62      AutoSeededRandomPool rng; //定义随机数发生器对象
63      ChannelSwitch *channelSwitch = new ChannelSwitch;
64      FileSource source(filename.c_str(), false, new SecretSharing(rng
            ,threshold, nShares,
65          channelSwitch )); //以filename文件为真实的Source构造数据分割链
66      //定义FileSink对象数组
67      vector_member_ptrs<FileSink> fileSinks(nShares);
68      std::string channel; //定义string对象，表示channel的ID
69      for (int i=0; i<nShares; i++)
70      {
71          //实现数据分割后文件名字的命名
72          char extension[5] = ".000";
73          extension[1]='0'+byte(i/100);
74          extension[2]='0'+byte((i/10)%10);
75          extension[3]='0'+byte(i%10);
76          fileSinks[i].reset(new FileSink((std::string(filename)+
                extension).c_str()));
77          //向channel对象添加数据链并完成ID的设置
78          channel = WordToString<word32>(i);
79          fileSinks[i]->Put((const byte *)channel.data(), 4);
80          channelSwitch->AddRoute(channel, *fileSinks[i],
                DEFAULT_CHANNEL);
81      }
82      source.PumpAll(); //将原始文件中的数据全部泵出
```

```
83 }
84 void SecretRecoverFile(int threshold,const string& outFilename,const
       vector<string>& inFilenames)
85 {
86     //nShares的范围为[1,1000]
87     CRYPTOPP_ASSERT(threshold >= 1 && threshold <=1000);
88     if (threshold < 1 || threshold > 1000) //份额输入不合法，则抛出异常
89         throw InvalidArgument("SecretRecoverFile: "
90             +IntToString(threshold)+" is not in range [1, 1000]");
91     //以输出文件为真实的Sink来构造SecretRecovery对象
92     SecretRecovery recovery(threshold, new FileSink(outFilename.
           c_str()));
93     //定义FileSource对象数组
94     vector_member_ptrs<FileSource> fileSources(threshold);
95     SecByteBlock channel(4); //存储SecByteBlock的ID
96     int i;
97     for (i=0; i<threshold; i++)
98     {
99         //重置对象
100        fileSources[i].reset(new FileSource(inFilenames[i].c_str(),
               false));
101        fileSources[i]->Pump(4); //泵出4个字节
102        fileSources[i]->Get(channel,4); //获取channel的ID
103        //完成数据链的附加
104        fileSources[i]->Attach(new ChannelSwitch(recovery,
105            std::string((char *)channel.begin(), 4)));
106    }
107    //依次泵出输入文件中的数据
108    while (fileSources[0]->Pump(256))
109        for (i=1; i<threshold; i++)
110            fileSources[i]->Pump(256);
111    for (i=0; i<threshold; i++)
112        fileSources[i]->PumpAll();
113 }
```

执行程序，并输入如下信息：

```
待分割文件的名字：密码模块安全要求.pdf
陷门值（threshold）:3
分割的份额数（nShares）:5
恢复文件时，拥有的陷门数：3
输入文件名：密码模块安全要求.pdf.000
输入文件名：密码模块安全要求.pdf.001
输入文件名：密码模块安全要求.pdf.004
设置恢复后的文件名：recover.pdf
请按任意键继续. . .
```

上面的输入示例表示：首先将“密码模块安全要求.pdf”文件分割成5个份额，且当有这些份额中的3个及其3个以上文件时，可以恢复出原始文件。接下来，使用分割后的3个文件（编号分别为000、001和004）来恢复原始文件，并恢复后的文件命名为“recover.pdf”。程序执行完毕后，在本程序所在的目录下可以看到分割原始文件产生的5个文件份额和从3个份额中恢复出来的原始文件，如图1所示。



名称	修改日期	类型	大小
Release	2018/11/28 21:54	文件夹	
CryptoPPRelease.props	2018/8/8 17:41	PROPS 文件	1 KB
recover.pdf	2018/11/28 21:58	Adobe Acrobat ...	1,432 KB
Secret Sharing.cpp	2018/11/28 21:36	C++ Source	5 KB
Secret Sharing.vcxproj	2018/11/26 15:08	VCXPROJ 文件	5 KB
Secret Sharing.vcxproj.fi...	2018/11/26 15:08	VC++ Project F...	1 KB
密码模块安全要求.pdf	2018/9/26 10:15	Adobe Acrobat ...	1,432 KB
密码模块安全要求.pdf.000	2018/11/28 21:57	000 文件	1,432 KB
密码模块安全要求.pdf.001	2018/11/28 21:57	WinRAR 压缩文件	1,432 KB
密码模块安全要求.pdf.002	2018/11/28 21:57	002 文件	1,432 KB
密码模块安全要求.pdf.003	2018/11/28 21:57	003 文件	1,432 KB
密码模块安全要求.pdf.004	2018/11/28 21:57	004 文件	1,432 KB

图1 执行文件分割和恢复操作产生的文件

12 Socket网络工具

下面的示例程序利用Pipeling范式技术将服务端中的某个文件发送到客户端。

12.1 服务端示例代码

下面是服务端程序示例代码。

```
#include<iostream> //使用cout、cin
#include<socketft.h> //使用Socket、SocketSink
#include<files.h> //使用FileSource
#include<string> //使用string
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()
{
    try
    {
        //服务端
        Socket::StartSockets(); //启动Socket相关服务
        Socket socServer; //定义一个Socket对象
        socServer.Create(SOCK_STREAM); //创建流式套接字
        unsigned int iport;
        string filename;
        cout << "请输入要绑定的端口号: " ;
        cin >> iport;
        cout << "请输入要发送的文件名字: " ;
        cin >> filename;
        socServer.Bind(iport); //绑定端口号
        socServer.Listen(); //监听客户端的连接请求
        Socket socClient; //定义一个Socket对象，保存连接的客户端套接字
        socServer.Accept(socClient); //接收客户端的连接
        //将文件发送到客户端socClient
        cout << "文件发送中..." << endl;
        FileSource fileSrc(filename.c_str(),true,new SocketSink(
            socClient));
        cout << "文件发送完毕..." << endl;
        socServer.CloseSocket(); //关闭服务端套接字
        socClient.CloseSocket(); //关闭客户端套接字
        Socket::ShutdownSockets(); //关闭Socket相关服务
    }
    catch(const Exception& e)
    {//出现异常
        cout << e.what() << endl; //异常原因
    }
    return 0;
}
```

12.2 客户端示例代码

下面是客户端程序示例代码。

```
#include<iostream> //使用cout、cin
#include<socketft.h> //使用Socket、SocketSource
#include<files.h> //使用FileSink
#include<string> //使用string
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()
{
    try
    {
        //客户端
        Socket::StartSockets(); //启动Socket相关服务
        Socket socClient; //定义一个Socket对象
        socClient.Create(SOCK_STREAM); //创建流式套接字
        string straddr; //存储IP地址
        unsigned int iport; //存储端口号
        cout << "请输入要连接主机的IP地址: " ;
        cin >> straddr;
        cout << "请输入要连接主机的端口号: " ;
        cin >> iport;
        //尝试连接指定IP地址和端口号的主机
        socClient.Connect(straddr.c_str(),iport);
        //接收服务端发送来的文件
        cout << "文件接收中..." << endl;
        SocketSource socSrc(socClient.GetSocket(),true,new FileSink(
            "receiver.txt"));
        cout << "文件接收完毕..." << endl;
        socClient.CloseSocket(); //关闭客户端套接字
        Socket::ShutdownSockets(); //关闭Socket相关服务
    }
    catch(const Exception& e)
    {//出现异常
        cout << e.what() << endl; //异常原因
    }
    return 0;
}
```

12.3 服务端和客户端程序运行结果

执行上面的程序，首先启动服务端程序，输入要绑定的端口号和将要发送的文件名字。然后启动客户端程序，输入要连接主机的IP地址和端口号，服务端和客户端的输入和输出信息分别如下：

服务端程序的输出信息：

```
请输入要绑定的端口号：5050
请输入要发送的文件名字：server_01.cpp
文件发送中...
文件发送完毕...
请按任意键继续. . .
```

客户端程序的输出信息：

```
请输入要连接主机的IP地址：127.0.0.1
请输入要连接主机的端口号：5050
文件接收中...
文件接收完毕...
请按任意键继续. . .
```

程序执行完毕后，在客户端程序所在的目录下会出现一个名为receiver.txt的文件，该文件就是由服务端发过来的server_01.cpp文件。

13 压缩工具

下面以Gzip类和Gunzip类为例，演示如何压缩和解压文件。

```
#include<iostream> //使用cout、cin
#include<files.h> //使用FileSource、FileSink
#include<gzip.h.> //使用Gzip、Gunzip
#include<string> //使用string
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()
{
    string filename;
    cout << "请输入待压缩文件的名字: " ;
    cin >> filename;
    //压缩文件
    FileSource fSrc1(filename.c_str(),true,new Gzip(new FileSink("
        compress.zip")));
    //解压缩文件
    FileSource fSrc2("compress.zip",true,new Gunzip(new FileSink("
        recover.txt")));
    return 0;
}
```

运行程序并输入一个名字为send.txt的文本文件，待程序执行完毕后，在程序所在目录下面可以看到一个名为compress.zip的压缩文件和一个名为recover.txt文件。前者是send.txt文件被压缩后的文件，后者是压缩文件compress.zip被解压后对应的文件。

14 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》

15 备注

原电子文档遗漏了《11 秘密分割工具》示例程序的输入范例，本文档对此进行了补充。