《第八章 分组密码》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 以CBC模式运行分组密码Camellia

下面以分组密码Camellia的CBC运行模式为例，演示如何用分组密码执行加密和解密等相关操作。

```
#include<iostream> //使用cout、cin
#include<camellia.h> //使用Camellia
#include<osrng.h> //使用AutoSeededRandomPool
#include<secblock.h> //使用SecByteBlock
#include<filters.h> //使用StringSource、StreamTransformationFilter
#include<hex.h> //使用HexEncoder
#include<files.h> //使用FileSink
#include<modes.h> //使用CBC_Mode
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()
{
    Camellia::Encryption cam; //定义加密对象
    cout << "缺省的密钥长度（字节）: "
        << cam.DefaultKeyLength() << endl;
    cout << "最小的密钥长度（字节）: "
        << cam.MinKeyLength() << endl;
    cout << "最大的密钥长度（字节）: "
        << cam.MaxKeyLength() << endl;
    cout << "分组长度（字节）: "
        << cam.BlockSize() << endl;
    //定义随机数发生器对象，用于产生密钥和初始向量
    AutoSeededRandomPool rng;
    SecByteBlock key; //用于存放密钥
    //待加密的明文字符串
    string plain = "I like cryptography very much.";
    //定义两个string对象，分别存储加密后的密文和解密后的明文
    string cipher, recover;
    SecByteBlock iv; //用于存放初始向量
    try
    {//加密
        CBC_Mode< Camellia >::Encryption enc; //定义加密器对象
        cout << "缺省的初始向量长度（字节）: "
            << enc.DefaultIVLength() << endl;
        cout << "最小的初始向量长度（字节）: "
            << enc.MinIVLength() << endl;
        cout << "最大的初始向量长度（字节）: "
            << enc.MaxIVLength() << endl;
        cout << "plain:"; //以十六进制打印输出待加密的明文
        StringSource sSrc(plain, true, new HexEncoder(new FileSink(
            cout)));
        key.resize(enc.DefaultIVLength()); //为key分配存储空间
```

```
        iv.resize(enc.DefaultIVLength()); //为iv分配存储空间
        rng.GenerateBlock(key, key.size()); //生成一个随机的密钥
        rng.GenerateBlock(iv, iv.size()); //产生初始向量iv
        //设置密钥和初始向量
        enc.SetKeyWithIV(key, key.size(), iv, iv.size());
        //加密字符串-利用enc加密字符串plain，并将加密结果存放于cipher中
        StringSource Enc(plain, true,
            new StreamTransformationFilter(enc,
                new StringSink(cipher)));
        cout << endl << "cipher:"; //以十六进制打印输出加密的结果，即密文
        StringSource sCipher(cipher, true,
            new HexEncoder(
                new FileSink(cout)));
    }
    catch (const Exception& e)
    {//出现异常
        cout << e.what() << endl; //异常原因
        return 0;
    }
    try
    {//解密
        CBC_Mode< Camellia >::Decryption dec; //定义解密器对象
        //设置密钥和初始向量
        dec.SetKeyWithIV(key, key.size(), iv, iv.size());
        //解密字符串-利用dec解密字符串cipher，并将解密结果存放于recover中
        StringSource Dec(cipher, true,
            new StreamTransformationFilter(dec,
                new StringSink(recover)));
        cout << endl << "recover:"; //以十六进制打印输出解密结果
        StringSource sRecover(recover, true,
            new HexEncoder(
                new FileSink(cout)));
        cout << endl;
  }
    catch (const Exception& e)
    {//出现异常
        cout << e.what() << endl; //异常原因
    }
    return 0;
}
```

执行程序，程序的输出结果如下：

```
缺省的密钥长度（字节）：16
最小的密钥长度（字节）：16
最大的密钥长度（字节）：32
```

```
分组长度（字节）：16
缺省的初始向量长度（字节）：16
最小的初始向量长度（字节）：16
最大的初始向量长度（字节）：16
plain:49206C696B652063727970746F6772617068792076657279206D7563682E
cipher:C463CC4251EDAFE798EFE964B7E07E9D3527B7F1E4371298B5AFC31F76CA5437
recover:49206C696B652063727970746F6772617068792076657279206D7563682E
请按任意键继续. . .
```

2 以EAX模式运行分组密码Camellia

下面仍以分组密码Camellia为例，演示认证性Filter的使用方法。

```
#include<iostream> //使用cout、cin
#include<camellia.h> //使用Camellia
#include<osrng.h> //使用AutoSeededRandomPool
#include<secblock.h> //使用SecByteBlock
//使用StringSource、AuthenticatedEncryptionFilter、AuthenticatedDecryptionFilter
#include<filters.h>
#include<hex.h> //使用HexEncoder
#include<files.h> //使用FileSink
#include<eax.h> //使用EAX
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()
{
    //定义随机数发生器对象，用于产生密钥和初始向量
    AutoSeededRandomPool rng;
    SecByteBlock key; //存储产生的密钥key
    //待加的密明文字符串
    string plain = "I like cryptography very much.";
    //定义两个string对象，分别存储加密后的密文和解密后的明文
    string cipher,recover;
    SecByteBlock iv; //存储初始向量
    try
    {//加密
        EAX< Camellia >::Encryption enc; //定义加密器对象
        key.resize(enc.DefaultKeyLength()); //申请一段空间，用于存放密钥
        rng.GenerateBlock(key,key.size()); //生成一个随机的密钥
        cout << "plain:" ; //以十六进制打印输出待加密的明文
        StringSource sSrc(plain,true,new HexEncoder (new FileSink(
            cout)));
        iv.resize(enc.DefaultIVLength()); //为iv分配存储空间
        rng.GenerateBlock(iv,iv.size()); //产生初始向量iv
        //设置密钥和初始向量
        enc.SetKeyWithIV(key,key.size(),iv,iv.size());
        //加密字符串-利用enc加密字符串plain，并将加密结果存放于cipher中
        StringSource Enc(plain,true,
            new AuthenticatedEncryptionFilter(enc,
                new StringSink(cipher)));
        //以十六进制打印输出加密的结果，即密文
        cout << endl << "cipher:" ;
        StringSource sCipher(cipher,true,
            new HexEncoder (
                new FileSink(cout)));
    }
```

```
43      catch(const Exception& e)
44      {//出现异常
45          cout << e.what() << endl; //异常原因
46          return 0;
47      }
48      try
49      {//解密
50          EAX< Camellia >::Decryption dec; //定义解密器对象
51          //设置密钥和初始向量
52          dec.SetKeyWithIV(key,key.size(),iv,iv.size());
53          //解密字符串-利用dec解密字符串cipher，并将解密结果存放于recover中
54          StringSource Dec(cipher,true,
55              new AuthenticatedDecryptionFilter(dec,
56                  new StringSink(recover)));
57          cout << endl << "recover:" ;
58          //以十六进制打印输出解密结果
59          StringSource sRecover(recover,true,
60              new HexEncoder (
61                  new FileSink(cout)));
62          cout << endl;
63      }
64      catch(const Exception& e)
65      {//出现异常
66          cout << e.what() << endl; //异常原因
67      }
68      return 0;
69  }
```

执行程序，程序的输出结果如下：

```
plain:49206C696B652063727970746F6772617068792076657279206D7563682E
cipher:B334AFF08DEAB4D250A8053536F01BD47FFFF2D1789641594161E6D3A88CB0AA23
3CBC20254D8A1137172BFB8E71
recover:49206C696B652063727970746F6772617068792076657279206D7563682E
请按任意键继续. . .
```

3 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》