《第九章 消息认证码》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 使用HMAC算法

下面以原书中图9.2的（c）所描述的过程为例，来演示加密、认证、验证和解密的具体方法。其中，用分组密码算法AES（以CBC模式运行）实现对消息的加密。利用Hash函数SHA3_512算法来实例化HMAC类模板并构造一个MAC算法，并用这个MAC算法认证加密后的消息。在本例中，我们使用Pipeling范式数据处理技术，示例程序的完整代码如下：

```
#include<iostream> //使用cout、cin
#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<string> //使用string
#include<hmac.h> //使用HMAC
#include<sha3.h> //使用SHA3_512
#include<aes.h> //使用AES
#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()
{
    //待加密和认证的消息
    string message = "I like cryptography very much.";
    string cipher,recover; //cipher存放加密后的密文，recover存放解密后的明文
    HMAC<SHA3_512> hmac; //利用HMAC框架和SHA3_521构造一种MAC算法
    //申请一段内存空间，存储hmac所使用的密钥
    SecByteBlock hkey(hmac.DefaultKeyLength());
    AutoSeededRandomPool rng; //定义一个随机数发生器对象
    rng.GenerateBlock(hkey,hkey.size()); //产生随机的密钥
    hmac.SetKey(hkey,hkey.size()); //设置MAC所使用的密钥
    //以CBC模式来运行分组密码
    //使用分组密码来加密消息，使用MAC来对消息进行认证
    //定义以CBC模式运行的AES加密对象
    CBC_Mode<AES>::Encryption cbc_aes_enc;
    //定义以CBC模式运行的AES解密对象
    CBC_Mode<AES>::Decryption cbc_aes_dec;
    //申请一段内存空间,存储分组密码使用的密钥key
    SecByteBlock aes_key(cbc_aes_enc.DefaultKeyLength());
    //申请一段内存空间,存储初始向量iv
    SecByteBlock aes_iv(cbc_aes_enc.DefaultIVLength());
    //产生随机的密钥key
    rng.GenerateBlock(aes_key,aes_key.size());
    //产生随机的初始向量iv
    rng.GenerateBlock(aes_iv,aes_iv.size());
    //设置加密对象和解密对象所使用的key和iv
    cbc_aes_enc.SetKeyWithIV(aes_key,aes_key.size(), aes_iv, aes_iv.
```

```
        size());
41      cbc_aes_dec.SetKeyWithIV(aes_key,aes_key.size(), aes_iv, aes_iv.
        size());
42      cout << "plain:"; //以十六进制的形式打印输出待加密消息message
43      StringSource plainSrc(message,true,
44          new HexEncoder(
45              new FileSink(cout)));
46      //先加密数据，再完成认证
47      //将加密和认证后的结果存储于cipher中
48      StringSource encSrc(message,true,
49          new StreamTransformationFilter(cbc_aes_enc, //完成加密
50              new HashFilter(hmac, //计算MAC值
51                  new StringSink(cipher), true)));
52      cout << endl << "cipher:";
53      //以十六进制的形式打印输出消息被加密和认证后的结果
54      StringSource cipherSrc(cipher,true,
55          new HexEncoder(
56              new FileSink(cout)));
57      //先验证，再完成解密
58      StringSource decSrc(cipher,true,
59          new HashVerificationFilter(hmac, //完成数据的验证
60              //完成数据的解密
61              new StreamTransformationFilter(cbc_aes_dec,
62                  new StringSink(recover)), //将解密的数据存储于recover中
63                  HashVerificationFilter::HASH_AT_END |
64                  HashVerificationFilter::PUT_MESSAGE |
65                  HashVerificationFilter::THROW_EXCEPTION));
66      cout << endl << "recover:"; //以十六进制打印输出解密后的明文
67      StringSource recoverSrc(recover,true,
68          new HexEncoder(
69              new FileSink(cout)));
70      cout << endl;
71      return 0;
72  }
```

执行程序，程序的输出结果如下：

```
plain:49206C696B652063727970746F6772617068792076657279206D7563682E
cipher:F11839008EEB1A5711416A6F15DFB4EE83269D7E5FF9152A82F1DE51FDBFE5B532
EFB091D37B6C210D3FEEB4F2D0D2E6C5EC73A20DA01671C5E2FCF2CE2DF07EA21C898
986BE275B9FBC257B4FDAF1676760D5CE8FEE50E2B9AFD4DC6744F2C3
recover:49206C696B652063727970746F6772617068792076657279206D7563682E
请按任意键继续. . .
```

2 利用Hash函数自定义消息认证码算法

下面演示如何实现原书中图9.3的方案（d）。在本示例中，我们以密码学Hash函数SM3做为MAC算法，以流密码SEAL做为加密算法。

```
#include<iostream> //使用cout、cin
//使用StringSource、StringSink、StreamTransformationFilter、ArraySource
#include<filters.h>
#include<files.h> //使用FileSink
#include<hex.h> //使用HexEncoder
#include<string> //使用string
#include<osrng.h> //使用AutoSeededRandomPool
#include<secblock.h> //使用SecByteBlock
#include<seal.h> //使用SEAL
#include<sm3.h> //使用SM3
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()
{
    //待加密和认证的消息
    string M="I like cryptography very much.";
    string cipher; //cipher存放加密后的密文
    cout << "plain:" ; //以十六进制打印输出待加密明文
    StringSource MSrc(M,true,
        new HexEncoder(
            new FileSink(cout)));
    //定义流密码SEAL加密对象
    SEAL<CryptoPP::BigEndian>::Encryption enc;
    //定义流密码SEAL对解密象
    SEAL<CryptoPP::BigEndian>::Decryption dec;
    SM3 H;//直接用Hash函数作为MAC算法
    //定义一个随机数发生器，用于产生随机的秘密信息
    AutoSeededRandomPool rng;
    //动态申请空间以存储接下来生成的密钥key和初始向量iv以及MAC算法的秘密信息
    SecByteBlock key(enc.DefaultKeyLength()),iv(enc.DefaultIVLength
        ());
    SecByteBlock S(64); //存储MAC算法的秘密信息
    //产生随机的key和iv
    rng.GenerateBlock(key,key.size());
    rng.GenerateBlock(iv,iv.size());
    rng.GenerateBlock(S,S.size());
    //设置加密和解密对象所使用的key和iv
    enc.SetKeyWithIV(key,key.size(),iv,iv.size());
    dec.SetKeyWithIV(key,key.size(),iv,iv.size());
    string tmp; //将秘密信息S转存至tmp中
    ArraySource arraySrc(S,S.size(),true,
        new StringSink(tmp));
```

```
    tmp = M + tmp; //将消息和秘密信息S进行连接
    string digest; //存储消息和秘密信息连接在一起的SM3消息摘要
    StringSource hashSrc(tmp, true,
        new HashFilter(H,
            new StringSink(digest)));
    tmp = M + digest; //将消息和计算的消息摘要连接在一起
    //加密连接在一起的消息和消息摘要
    StringSource encSrc(tmp, true,
        new StreamTransformationFilter(enc,
            new StringSink(cipher)));
    cout << endl << "cipher:" ; //以十六进制打印输出密文
    StringSource cipherSrc(cipher, true,
        new HexEncoder(
            new FileSink(cout)));
    tmp.clear(); //清空tmp
    //解密连接在一起的消息和消息摘要的密文
    StringSource decSrc(cipher, true,
        new StreamTransformationFilter(dec,
            new StringSink(tmp)));
    //将tmp表示的明文和消息摘要分离
    //得到消息摘要，存储于dig
    string dig(tmp.begin()+tmp.length()-H.DigestSize(),tmp.end());
    //recover存放解密后的明文
    string recover(tmp.begin(),tmp.begin()+tmp.length()-H.DigestSize
        ());
    tmp.clear(); //清空tmp
    //将消息recover和秘密信息连接在一起
    ArraySource arrSrc(S,S.size(),true,
        new StringSink(tmp));
    tmp = recover + tmp;
    //计算消息摘要并比较
    bool bmatch = H.VerifyDigest((CryptoPP::byte*)dig.c_str(),
        (CryptoPP::byte*)tmp.c_str(),tmp.length());
    if(bmatch)
    {//验证成功
        cout << endl << "recover:" ; //以十六进制打印输出解密后的明文
        StringSource recoverSrc(recover,true,
            new HexEncoder(
                new FileSink(cout)));
    }
    else
    {//验证失败
        cout << endl << "验证失败！" << endl;
    }
    cout << endl ;
    return 0;
```

```
87 }
```

执行程序，程序的输出结果如下：

```
plain:49206C696B652063727970746F6772617068792076657279206D7563682E
cipher:507C389E892AB58E83C98BE77A731C1CB42435F4E15F296B63F5233268780BC8D292
E6932A2A89F9DD10B39814BEE56023D8037B6E4E04C279C1255057C0
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》