《第六章 Hash函数》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 计算字符串的Hash值

本示例演示使用Hash函数计算消息摘要的方法。

```
#include<iostream> //使用cout、cin
#include<sha.h> //使用SHA384
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
    {
        SHA384 sha; //定义一个SHA384的类对象
        //分组长度（比特）
        cout << "BlockSize=" << sha.BlockSize()*8 << endl;
        //Hash值长度（比特）
        cout << "DigestSize=" << sha.DigestSize()*8 << endl;
        cout << "TagSize=" << sha.TagSize()*8 << endl; //等于DigestSize
        //最优分组大小（比特）
        cout << "OptimalBlockSize=" << sha.OptimalBlockSize()*8 <<
            endl;
        //最优输入输出数据对齐大小（字节）
        cout << "OptimalDataAlignment=" << sha.OptimalDataAlignment
            () << endl;
        CryptoPP::byte msg[] = "I like cryptography very much";
        CryptoPP::byte msg1[] = "I like cryptography "; //最后有空格
        CryptoPP::byte msg2[] = "very much";
        //先计算msg1+masg2拼接后的消息的Hash值
        //向Hash函数输入消息msg1
        sha.Update(msg1,sizeof(msg1)-1); //去掉字符串最后的'\0'
        //向Hash函数输入消息msg2
        sha.Update(msg2,sizeof(msg2)-1); //去掉字符串最后的'\0'
        //求拼接后的消息"I like cryptography very much"的Hash值
        size_t len1=sha.DigestSize();
        //申请内存空间以存放消息摘要
        CryptoPP::byte* digest1=sha.CreateUpdateSpace(len1);
        if(len1 < sha.DigestSize())
        {
            cout << "分配的内存不足" << endl;
            return 0;
        }
        //计算Hash值，同时重置Hash函数内部状态
        sha.Final(digest1);
        cout << "digest1=";
        for(size_t i=0; i < sha.DigestSize(); ++i)
        {//以十六进制输出Hash值
            printf("%02X",digest1[i]);
```

```
        }
        cout << endl;
        //申请内存空间以存放消息摘要
        SecByteBlock digest2(sha.DigestSize());
        //CalculateDigest()相当于Update()+Final()
        sha.CalculateDigest(digest2,msg,sizeof(msg)-1);
        cout << "digest2=";
        for(size_t i=0; i < sha.DigestSize(); ++i)
        {//以十六进制输出Hash值
            printf("%02X",digest2[i]);
        }
        cout << endl;
        //计算msg消息的Hash值
        bool res;
        res=sha.VerifyDigest(digest2, //可能抛出异常
        msg,sizeof(msg)-1); //去掉字符串最后的'\0'
        cout << "res = " << boolalpha << res << endl;
        delete[] digest1;//释放内存
    }
    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：

```
BlockSize=1024
DigestSize=384
TagSize=384
OptimalBlockSize=1024
OptimalDataAlignment=8
digest1=AA2DCB5FB010CDFE5FE83075EB219DA9B4AD9432D4DD06951171462F35ED13C
96326643033FAA4C7A82FA2F8C6BAF2D1
digest2=AA2DCB5FB010CDFE5FE83075EB219DA9B4AD9432D4DD06951171462F35ED13C
96326643033FAA4C7A82FA2F8C6BAF2D1
res = true
请按任意键继续. . .
```

2 计算文件的Hash值

本示例程序以CryptoPP54.zip源代码包为例，演示如何用Hash函数计算文件的Hash值。

```
#define _CRT_SECURE_NO_WARNINGS
#include<sha.h> //使用SHA1、SHA256、SHA512
#include<string> //使用string
#include<whrlpool.h> //使用Whirlpool
#include<iostream> //使用cout、cin
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：用Hash函数求文件CryptoPP54.zip的消息摘要，
//  并在标准输出设备上打印计算的结果
//参数name：打印输出计算的Hash的前缀信息
//参数hash：一个Hash函数对象的引用
void PerformHash(const string& name, HashTransformation& hash);
int main()
{
    SHA1   sha1; //定义一个SHA1对象
    PerformHash("SHA1",sha1); //计算文件的Hash值
    SHA256 sha256; //定义一个SHA256对象
    PerformHash("SHA256",sha256); //计算文件的Hash值
    SHA512 sha512; //定义一个SHA512对象
    PerformHash("SHA512",sha512); //计算文件的Hash值
    Whirlpool whpool; //定义一个Whirlpool对象
    PerformHash("Whirlpool",whpool); //计算文件的Hash值
    return 0;
}
void PerformHash(const string& name, HashTransformation& hash)
{
    cout << name<<":"<<endl;
    FILE*  fp; //定义文件指针，也可使用C++的文件操作方法
    fp = fopen("CryptoPP54.zip","rb"); //打开文件
    if(!fp)
    {
        cout << "打开文件失败" << endl;
        return ;
    }
    SecByteBlock digest(hash.DigestSize()); //存放计算的Hash值
    CryptoPP::byte buffer[1024]; //定义缓冲区buffer
    int len; //用于存储从文件中实际读取的数据长度
    while(!feof(fp) && !ferror(fp)) //判断文件流当前的状态
    {
        //从文件中读取数据
        len = fread(buffer,sizeof(CryptoPP::byte),1024,fp);
        //用读取的数据更新Hash函数
        hash.Update(buffer,(len < 1024 ? len:1024 ));
```

```
44      }
45      hash.Final(digest); //计算Hash值
46      for(int i=0; i < hash.DigestSize();++i)
47      {//以十六进制的形式输出Hash值
48          printf("%02X",digest[i]); //也可以使用C++的输出操作方式
49      }
50      cout << endl;
51      fclose(fp); //关闭文件
52  }
```

执行程序，程序的输出结果如下：

```
SHA1:
88F6534B713FBBF5C1AF5FDDDC402B221EEA73BF
SHA256:
FA9ACEB1B46C886B5C13FE5AA3D0CDBD74B4A2DD894E290CBDBFD17FE8A7FE5A
SHA512:
C97AD75D0240F6CD907267CACFBCDF3B6EDC22412DF6A0C9CEDAB7B1159C44BB601F
880F1F147462350BF5B17796421CDE40DD444BBDD31CAC848087429A434D
Whirlpool:
9B83F4A1CFD7F49CE5E123169D0AB291929D5D3B82355F22BEDA39DF47E1035B53AA5D
1A20A5662214E9BF4A07028792D219EEDF7C888D45BF653181AECEC839
请按任意键继续. . .
```

3 以Pipeling范式方式使用Hash函数

下面以Pipeling范式技术实现前一个示例类似的功能。

```
#include<iostream> //使用cout、cin
#include<channels.h> //使用ChannelSwich
#include<string> //使用string
#include<files.h> //使用FileSource
#include<sha.h> //使用SHA1、SHA256、SHA512
#include<filters.h> //使用HashFilter、Redirector
#include<hex.h> //使用HexEncoder
#include<whrlpool.h> //使用Whirlpool
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 = "CryptoPP54.zip";
    string s1,s2,s3,s4; //定义4个string对象分别用于存放求得的Hash值
    SHA1 sha; //定义SHA1对象
    SHA256 sha256; //定义SHA256对象
    SHA512 sha512; //定义SHA512对象
    Whirlpool whipool; //定义Whirlpool对象
    //定义HashFilter对象
    HashFilter f1(sha,new HexEncoder(new StringSink(s1)));
    //定义HashFilter对象
    HashFilter f2(sha256,new HexEncoder( new StringSink(s2)));
    //定义HashFilter对象
    HashFilter f3(sha512,new HexEncoder( new StringSink(s3)));
    //定义HashFilter对象
    HashFilter f4(whipool,new HexEncoder( new StringSink(s4)));
    ChannelSwitch cs; //定义ChannelSwitch对象
    cs.AddDefaultRoute(f1); //添加第1条数据链
    cs.AddDefaultRoute(f2); //添加第2条数据链
    cs.AddDefaultRoute(f3); //添加第3条数据链
    cs.AddDefaultRoute(f4); //添加第4条数据链
    //让filename表示的文件中的数据分别流向上述4条数据链
    FileSource ss(filename.c_str(),true,new Redirector(cs));
    //此时，s1,s2,s3,s4中分别存放了计算的相应Hash值
    cout << "SHA1:" << s1 << endl; //打印输出
    cout << "SHA256:" << s2 << endl; //打印输出
    cout << "SHA512:" << s3 << endl; //打印输出
    cout << "Whirlpool:" << s4 << endl; //打印输出
  return 0;
}
```

执行程序，程序的输出结果如下（与前一个示例相同）：

```
SHA1:
88F6534B713FBBF5C1AF5FDDDC402B221EEA73BF
SHA256:
FA9ACEB1B46C886B5C13FE5AA3D0CDBD74B4A2DD894E290CBDBFD17FE8A7FE5A
SHA512:
C97AD75D0240F6CD907267CACFBCDF3B6EDC22412DF6A0C9CEDAB7B1159C44BB601F
880F1F147462350BF5B17796421CDE40DD444BBDD31CAC848087429A434D
Whirlpool:
9B83F4A1CFD7F49CE5E123169D0AB291929D5D3B82355F22BEDA39DF47E1035B53AA5D
1A20A5662214E9BF4A07028792D219EEDF7C888D45BF653181AECEC839
请按任意键继续. . .
```

4 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》