《第十三章 数字签名》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RWSS数字签名算法

下面以RWSS算法为例，演示CryptoPP库中数字签名算法的使用方法。它需要用户指定签名标准和Hash函数类型。

```
#include<iostream>//使用cout、cin
#include<osrng.h>//使用AutoSeededRandomPool
#include<rw.h>//使用RWSS等
#include<sha3.h>//使用SHA3_384
#include<pssr.h>//使用PSSR
#include<hex.h>//使用HexEncoder
#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
    {
        AutoSeededRandomPool rng;//定义一个随机数发生器对象
        RW::PrivateKey prikey;//定义私钥对象
        prikey.GenerateRandomWithKeySize(rng,1024);
        RW::PublicKey pubkey(prikey);//定义公钥对象
        cout << "prikey:" ;
        //以十六进制输出私钥至标准输出设备
        prikey.Save(HexEncoder(new FileSink(cout)).Ref());
        cout << endl << "pubkey:" ;
        //以十六进制输出公钥至标准输出设备
        pubkey.Save(HexEncoder(new FileSink(cout)).Ref());
        cout << endl;
        if(prikey.Validate(rng,3))
            cout << "私钥满足要求的安全性级别" << endl;
        else
            cout << "私钥不满足要求的安全性级别" << endl;
        if(pubkey.Validate(rng,3))
            cout << "公钥满足要求的安全性级别" << endl;
        else
            cout << "公钥不满足要求的安全性级别" << endl;
        string message="I like Cryptography.";//待签名的消息
        cout << "message:" ;
        //以十六进制输出消息message至标准输出设备
        StringSource messageSrc(message,true,
            new HexEncoder(
                new FileSink(cout)));
        string signature;//存储被签名后的消息strSign
        string recover;//存储从签名中恢复出的消息
        RWSS<PSSR,SHA3_384>::Signer Sig(prikey);//定义签名对象
```

```
        RWSS<PSSR, SHA3_384>::Verifier Ver(pubkey);//定义验证对象
        //执行签名-对字符串message签名，将签名的结果存储于signature
        StringSource SigSrc(message, true,
            new SignerFilter(rng, Sig,
                new StringSink(signature)));
        cout << endl << "signature:" ;
        //以十六进制输出签名Signature至标准输出设备
        StringSource SignSrc(signature, true,
            new HexEncoder(
                new FileSink(cout)));
        //执行签名的验证-输入消息+签名，验证该签名是否正确，
        //如果验证成功，将消息输出至recover;否则，抛出异常。
        StringSource VerSrc(message+signature, true,
            new SignatureVerificationFilter(Ver,
                new StringSink(recover),
                    SignatureVerificationFilter::PUT_MESSAGE |
                    SignatureVerificationFilter::THROW_EXCEPTION));
        cout << endl << "recover:" ;
        //以十六进制输出消息recover至标准输出设备
        StringSource strRecoverSrc(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：

```
prikey:3082014C02818100A446E008C13B74B71CC354ACFB81A6B85A4E80ABD4C105E451D
C59D7C7B8F457BB6DE0877819533556666EDF3D63D970C89F1BD231A554F773732946AB42
8DC69DF76B9837281593B5E9F451ABD9CA5368D57DFBE6050B4ED39D30795215F13566E3
0B09261C60B8129A2D9561ADE412938EF90CEEBF7E4AB6EFF94A186D8ED5024100C9A1F
B661E86F03EEE545D73E7A43EB439D5355A142FC116298C8A86CA662D9AE0C5B6735A5C
243B2FF5237DD8C28B68320EFC8FACB6676FF33B7D2305D25663024100D092581A50647EC
8251B7F58BEB1411FE4FF499E5649798DD14481E662408A0632199276D135E37DA3FD9F810
BB5039B32C2F859E38AC98ADA08316FD55F0F6702406137EC7199090197DFEBD4FAEC696
88D0FC2F8D6E2671068BDF1D90F65CF42127F1C8AC9B4445F75E67A5B0FDB230444A7D6
1FBCD8E3A30B8CEAA4E08EF6CC37
pubkey:30818402818100A446E008C13B74B71CC354ACFB81A6B85A4E80ABD4C105E451DC
59D7C7B8F457BB6DE0877819533556666EDF3D63D970C89F1BD231A554F773732946AB428
DC69DF76B9837281593B5E9F451ABD9CA5368D57DFBE6050B4ED39D30795215F13566E30
```

```
B09261C60B8129A2D9561ADE412938EF90CEEBF7E4AB6EFF94A186D8ED5
私钥满足要求的安全性级别
公钥满足要求的安全性级别
message:49206C696B652043727970746F6772617068792E
signature:2A6BB783ADF4ECEB8FBAC45F37AA820E6CC38520300C884B60041772357C2739
B6FA4A8DD4FD1DC1AAF35EFCC42DA43D3D827CC0552FC6F4AF5C4ACCE891FF3DD2
C26A19A8F8F733928DB6A2EA26E7EF402DE9761EBCCBA43E707D269A4BF7808E647CB7
547BE845876F41573903BC7B7388A61793C29D9DD265B860B6E287A0
recover:49206C696B652043727970746F6772617068792E
请按任意键继续. . .
```

2 使用ECNR数字签名算法

下面以ECNR签名算法为例，演示CryptoPP库中另一类签名算法的使用方法。它需要用户指定具体的代数结构和Hash函数类型。

```
#include<iostream>//使用cout、cin
#include<osrng.h>//使用AutoSeededRandomPool
#include<eccrypto.h>//使用ECNR等
#include<oids.h>//使用ASN1::secp160r1
#include<sha3.h>//使用SHA256
#include<hex.h>//使用HexEncoder
#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
    {
        AutoSeededRandomPool rng;//定义一个随机数发生器对象
        ECNR<ECP,SHA256>::PrivateKey prikey;//定义ECNR私钥对象
        ECNR<ECP,SHA256>::PublicKey pubkey;//定义ECNR公钥对象
        //使用标准的椭圆曲线参数
        prikey.Initialize(rng,ASN1::secp160r1());
        //如果私钥不满足要求的安全级别，那么将引发断言
        CRYPTOPP_ASSERT(prikey.Validate(rng,3));
        prikey.MakePublicKey(pubkey);//根据私钥产生对应的公钥
        //如果公钥不满足要求的安全级别，那么将引发断言
        CRYPTOPP_ASSERT(pubkey.Validate(rng,3));
        string message="I like Cryptography.";//待签名的消息
        cout << "message:" ;
        //以十六进制输出消息message至标准输出设备
        StringSource messageSrc(message,true,
            new HexEncoder(
                new FileSink(cout)));
        string signature;//存储被签名后的消息strSign
        string recover;//存储从签名中恢复出的消息
        ECNR<ECP,SHA256>::Signer Sig(prikey);//定义签名器对象
        ECNR<ECP,SHA256>::Verifier Ver(pubkey);//定义验证器对象
        //执行签名–对字符串message签名，将签名的结果存储于Signature
        StringSource SigSrc(message,true,
            new SignerFilter(rng,Sig,
                new StringSink(signature),true));
        cout << endl << "signature:" ;
        //以十六进制输出签名Signature至标准输出设备
        StringSource SignSrc(signature,true,
            new HexEncoder(
```

```
                    new FileSink(cout)));
        //执行签名的验证–签名，验证该签名是否正确，
        //如果验证成功，将消息输出至Recover;否则，抛出异常。
        StringSource VerSrc(signature,true,
            new SignatureVerificationFilter(Ver,
                new StringSink(recover),
                    SignatureVerificationFilter::PUT_MESSAGE |
                    SignatureVerificationFilter::THROW_EXCEPTION));
        cout << endl << "recover:";
        //以十六进制输出消息recover至标准输出设备
        StringSource strRecoverSrc(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：

```
message:49206C696B652043727970746F6772617068792E
signature:49206C696B652043727970746F6772617068792E0034EFAB15AA687A0B32F444B629
D51B2EC9D6E2270046A323C303838D47683F371A44C595E222B97BFB
recover:49206C696B652043727970746F6772617068792E
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》