

《第十章 密钥派生和基于口令的密码》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、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	使用密钥派生函数HKDF	1
2	利用基于口令的密钥派生函数实现数据保护	3
2.1	基于口令的数据保护端程序	3
2.2	基于口令的数据访问端程序	7
3	声明	10

1 使用密钥派生函数HKDF

下面以密钥派生函数HKDF为例，演示CryptoPP库中密钥派生类算法的使用方法。

```
1 #include<iostream> //使用cout、cin
2 #include<hkdf.h> //使用HKDF
3 #include<osrng.h> //使用AutoSeededRandomPool
4 #include<sha3.h> //使用SHA3_512
5 #include<filters.h> //使用ArraySink
6 #include<hex.h> //使用HexEncoder
7 #include<files.h> //使用FileSink
8 #include<secblock.h> //使用SecByteBlock
9 using namespace std; //std是C++的命名空间
10 using namespace CryptoPP; //CryptoPP是CryptoPP库的命名空间
11 int main()
12 {
13     //定义一个随机数发生器，用于产生一些随机信息
14     AutoSeededRandomPool rng;
15     HKDF<SHA3_512> hkdf; //用SHA3_512实例化一个密钥派生函数
16     //申请一些存储空间，存储生成的随机信息
17     SecByteBlock salt(16); //16字节的盐值
18     SecByteBlock info(16); //16字节的附加信息
19     SecByteBlock secret(32); //32字节的秘密信息
20     //利用随机数发生器产生这些随机信息
21     rng.GenerateBlock(salt, salt.size()); //产生盐值
22     rng.GenerateBlock(info, info.size()); //产生附加信息
23     rng.GenerateBlock(secret, secret.size()); //产生秘密信息
24     SecByteBlock derived_key(128); //存储派生的密钥（长度为128字节）
25     //派生密钥
26     hkdf.DeriveKey(derived_key, derived_key.size(),
27         secret, secret.size(),
28         salt, salt.size(),
29         info, info.size());
30     cout << "最大的密钥派生长度："
31         << hkdf.MaxDerivedLength() << endl;
32     cout << "最小的密钥派生长度："
33         << hkdf.MinDerivedLength() << endl;
34     cout << "有效的密钥派生长度："
35         << hkdf.GetValidDerivedLength(derived_key.size()) << endl;
36     cout << "salt:" ;//输出打印salt
37     //以十六进制的形式打印显示salt
38     ArraySource saltSrc(salt, salt.size(), true,
39         new HexEncoder(
40             new FileSink(cout)));
41     cout << endl << "info:" ;//输出打印info
42     //以十六进制的形式打印显示info
43     ArraySource infoSrc(info, info.size(), true,
```

```

44     new HexEncoder(
45         new FileSink(cout));
46     cout << endl << "secret:" ; //输出打印secret
47     //以十六进制的形式打印显示secret
48     ArraySource secretSrc(secret, secret.size(), true,
49         new HexEncoder(
50             new FileSink(cout)));
51     cout << endl << "derived_key:" ; //输出打印derived_key
52     //以十六进制的形式打印显示derived_key
53     ArraySource derived_keySrc(derived_key, derived_key.size(), true,
54         new HexEncoder(
55             new FileSink(cout)));
56     cout << endl;
57     return 0;
58 }

```

执行程序，程序的输出结果如下：

```

最大的密钥派生长度：16320
最小的密钥派生长度：0
有效的密钥派生长度：128
salt:7179A5DA758CD07BF2D87EB9F1EE0555
info:D08DE3AA101A8412E776B8079283DDA7
secret:4BE1EACB3E8FAAF449F08CDECF48C5F4AF6032AFF01F5D7DDA8A66A4C220928A
derived_key:C80E35F6AC28726B17AFC52953C1102E62441F99D880FA93AD7368B9554F79073
FB8BF4E2AA4DD09498A13C88F0290581EBCAEF52C3E881428A06030CE2E1CD8FF016152
D0958237B44D79EED77472EA939464317C69B34672CA2C3D0C096936B31F3B8C0EDC54B4
C35BE9D5377AE9D5804062DD2C435F112F356650B3F50D2C
请按任意键继续...

```

2 利用基于口令的密钥派生函数实现数据保护

下面以基于口令的密钥派生算法PKCS12_PBKDF为例，演示如何通过口令实现数据保护。在本程序中，待保护或访问的数据是磁盘上的一个文件，它的基本原理如原书中图10.3的(c)所示。为了更好地向读者演示基于口令的数据保护过程，我们将程序分为两部分，即基于口令的数据保护端和基于口令的数据访问端，它们的执行原理分别如原书中图10.7和10.9所示。

2.1 基于口令的数据保护端程序

下面使用基于口令的密钥派生算法实现protecting_data.txt文件的保护，完整示例代码如下：

```
1 #include<iostream> //使用cout、cin
2 #include<osrng.h> //使用AutoSeededRandomPool
3 #include<secblock.h> //使用SecByteBlock
4 #include<gcm.h> //使用GCM
5 #include<eax.h> //使用EAX
6 #include<aes.h> //使用AES
7 #include<sm4.h> //使用SM4
8 #include<ripemd.h> //RIPEMD320
9 #include<pwdbased.h> //PKCS12_PBKDF
10 #include<string> //使用string
11 //使用AuthenticatedEncryptionFilter、AuthenticatedDecryptionFilter
12 #include<filters.h>
13 #include<files.h>> //使用FileSink、FileSource
14 #include<hex.h> //使用HexEncoder
15 using namespace std; //使用C++标准命名空间std
16 using namespace CryptoPP; //CryptoPP是CryptoPP库的命名空间
17 int main()
18 {
19     try
20     {
21         string password; //存储用户输入的口令
22         cout << "请输入保护数据的口令（不少于10个字符）：" << endl;
23         getline(cin, password); //用户输入口令
24         AutoSeededRandomPool rng; //定义随机数发生器对象
25         SecByteBlock salt(128); //存储盐值，大小为128*8=1024（比特）
26         size_t count = 100000; //迭代次数，大小为10万次
27         EAX<SM4>::Encryption eax_sm4_enc; //定义保护数据的加密器对象
28         GCM<AES>::Encryption gcm_aes_enc; //定义保护DPK的加密器对象
29         size_t dpk_len = eax_sm4_enc.DefaultKeyLength() +
30             eax_sm4_enc.DefaultIVLength();
31         //产生dpk_len长度的数据，用作SM4算法的密钥和初始向量
32         SecByteBlock DPK(dpk_len);
33         size_t mk_len = gcm_aes_enc.DefaultKeyLength() + gcm_aes_enc
34             .DefaultIVLength();
```

```

33 //产生mk_len长度的主密钥，用作AES算法的密钥和初始向量
34 SecByteBlock MK(mk_len);
35 // (1) 产生MK
36 //利用随机数发生器产生随机的盐值
37 rng.GenerateBlock(salt, salt.size());
38 PKCS12_PBKDF<RIPEMD320> pbkdf; //定义基于口令的密钥派生函数对象
39 pbkdf.DeriveKey(MK, MK.size(), //存储派生的主密钥
40     static_cast<byte>('M'), //目的前缀
41     //口令（秘密信息）
42     (CryptoPP::byte*)password.c_str(), password.size(),
43     salt, salt.size(), //盐值
44     count, //迭代次数
45     0.0); //运行时间
46 cout << "salt: "; //以十六进制打印输出盐值salt
47 ArraySource salt_Src(salt, salt.size(), true,
48     new HexEncoder(
49         new FileSink(cout)));
50 cout << endl;
51 // (2) 产生DPK
52 //利用随机数发生器产生随机的DPK
53 rng.GenerateBlock(DPK, DPK.size());
54 string dpk_enc; //存储被加密的DPK密文
55 //设置key和iv
56 gcm_aes_enc.SetKeyWithIV(MK, gcm_aes_enc.DefaultKeyLength(),
57     MK + gcm_aes_enc.DefaultKeyLength(),
58     gcm_aes_enc.DefaultIVLength());
59 //加密并且认证
60 //将加密认证的结果存储于dpk_enc
61 ArraySource enc_dpk_Src(DPK, DPK.size(), true,
62     new AuthenticatedEncryptionFilter(gcm_aes_enc,
63     new StringSink(dpk_enc)));
64 cout << "dpk_enc: "; //以十六进行打印输出被加密和认证的DPK
65 StringSource dpk_enc_Src(dpk_enc, true,
66     new HexEncoder(
67         new FileSink(cout)));
68 cout << endl;
69 //将盐值salt和加密认证后的DPK存储到磁盘上，供用户访问数据时使用
70 FileSink fSink("salt_dpkenct.txt");
71 fSink.Put(salt, salt.size()); //先将盐值存入文件
72 //再将加密认证后的DPK存入文件
73 fSink.Put((CryptoPP::byte*)dpk_enc.c_str(), dpk_enc.length()
74     );
75 fSink.MessageEnd(); //关闭文件
76 // (3) 保护文件
77 //设置key和iv
78 eax_sm4_enc.SetKeyWithIV(DPK,

```

```

78     eax_sm4_enc.DefaultKeyLength(),
79     DPK + eax_sm4_enc.DefaultKeyLength(),
80     eax_sm4_enc.DefaultIVLength());
81     //protecting_data.txt-待加密和认证的文件
82     //执行加密和认证
83     //将密文存储于protected_data.txt文件
84     FileSource enc_file_Src("protecting_data.txt", true,
85         new AuthenticatedEncryptionFilter(eax_sm4_enc,
86             new FileSink("protected_data.txt")));
87 }
88 catch (const Exception& e)
89 { //出现异常
90     cout << e.what() << endl; //异常原因
91 }
92 return 0;
93 }

```

执行程序，并输入一个口令（hanlulu1234567890cryptopp），运行结果如图1所示。

```

C:\windows\system32\cmd.exe
请输入保护数据的口令（不少于10个字符）：
hanlulu1234567890cryptopp
salt: 5E81694F5930DC2727AC4C46E7530AF218DCEDBFE6D29769EB9D40A98366C8778CE6B7E72A
25CAF07147016B2EB8BD60575C422533BBA3B0D0AB35186583B41C847719346C445022BEB4EC1920
CAD5579DDEF286899A2359D98B5868AA7731239E7B5CCBB86C011595F20C373578872BD5CC1736D
90435289FDD563026F7008
dpk_enc: F1AC6A7D818F487E4FAA335FE8DB24CC5C85F7C987C782F77733CB2637DE2F1B1AC16BB
4C2ABE6F63A2AB35EB005BA74
请按任意键继续. . .

```

图1 数据保护端程序运行结果

运行数据保护端程序，当程序执行完毕后，在该程序所在的目录下会产生两个文件，它们分别是salt_dpkenc.txt和protected_data.txt。将这两个文件分别复制到数据访问端程序所在

目录，执行数据访问端程序并输入口令，即可从文件protected_data.txt中恢复出被加密的原始明文。

2.2 基于口令的数据访问端程序

下面使用基于口令的密钥派生算法实现对protected_data.txt文件的访问，完整示例代码如下：

```
1 #include<iostream> //使用cout、cin
2 #include<osrng.h> //使用AutoSeededRandomPool
3 #include<secblock.h> //使用SecByteBlock
4 #include<gcm.h> //使用GCM
5 #include<eax.h> //使用EAX
6 #include<aes.h> //使用AES
7 #include<sm4.h> //使用SM4
8 #include<ripemd.h> //RIPEMD320
9 #include<pwdbased.h> //PKCS12_PBKDF
10 #include<string> //使用string
11 //使用AuthenticatedEncryptionFilter、AuthenticatedDecryptionFilter
12 #include<filters.h>
13 #include<files.h>> //使用FileSink、FileSource
14 #include<hex.h> //使用HexEncoder
15 using namespace std; //使用C++标准命名空间std
16 using namespace CryptoPP; //CryptoPP是CryptoPP库的命名空间
17 int main()
18 {
19     try
20     {
21         string password; //存储用户输入的口令
22         cout << "请输入访问数据的口令（不少于10个字符）：" << endl;
23         getline(cin, password); //用户输入口令
24         SecByteBlock salt(128); //存储盐值，大小为128*8=1024（比特）
25         size_t count = 100000; //迭代次数，大小为10万次
26         //定义访问数据的解密器对象
27         EAX<SM4>::Decryption eax_sm4_dec;
28         //定义访问DPK的解密器对象
29         GCM<AES>::Decryption gcm_aes_dec;
30         size_t dpk_len = eax_sm4_dec.DefaultKeyLength() +
            eax_sm4_dec.DefaultIVLength();
31         //产生dpk_len长度的数据，用作SM4算法的密钥和初始向量
32         SecByteBlock DPK(dpk_len);
33         size_t mk_len = gcm_aes_dec.DefaultKeyLength() + gcm_aes_dec
            .DefaultIVLength();
34         //产生mk_len长度的主密钥，用作AES算法的密钥和初始向量
35         SecByteBlock MK(mk_len);
36         //（1）重构MK
37         //读取文件中的数据，并存储于tmp对象中
38         string tmp; //存储从文件读取的盐值和被加密的DPK
39         FileSource fource("salt_dpkenr.txt", true,
40             new StringSink(tmp)); //读取数据
```

```

41 memcpy(salt, tmp.c_str(), salt.size());
42 cout << "salt: "; //以十六进制打印输出盐值salt
43 ArraySource salt_Src(salt, salt.size(), true,
44     new HexEncoder(
45         new FileSink(cout)));
46 cout << endl;
47 PKCS12_PBKDF<RIPEMD320> pbkdf; //定义基于口令的密钥派生函数对象
48 pbkdf.DeriveKey(MK, MK.size(), //存储派生的主密钥
49     static_cast<byte>('M'), //目的前缀
50     //口令（秘密信息）
51     (CryptoPP::byte*)password.c_str(), password.size(),
52     salt, salt.size(), //盐值
53     count, //迭代次数
54     0.0); //运行时间
55 // (2) 恢复DPK
56 //存储被加密的DPK密文
57 SecByteBlock dpk_enc(tmp.length()-salt.size());
58 memcpy(dpk_enc, tmp.c_str() + salt.size(), dpk_enc.size());
59 //设置key和iv
60 gcm_aes_dec.SetKeyWithIV(MK, gcm_aes_dec.DefaultKeyLength(),
61     MK + gcm_aes_dec.DefaultKeyLength(),
62     gcm_aes_dec.DefaultIVLength());
63 cout << "dpk_enc: "; //以十六进行打印输出被加密和认证的DPK
64 ArraySource dpk_enc_Src(dpk_enc, dpk_enc.size(), true,
65     new HexEncoder(
66         new FileSink(cout)));
67 cout << endl;
68 //验证并解密
69 //将解密的结果存储于DPK
70 ArraySource enc_dpk_Src(dpk_enc, dpk_enc.size(), true,
71     new AuthenticatedDecryptionFilter(gcm_aes_dec,
72     new ArraySink(DPK,DPK.size())));
73 // (3) 访问文件
74 string recover; //存储文件的明文内容
75 //设置key和iv
76 eax_sm4_dec.SetKeyWithIV(DPK,
77     eax_sm4_dec.DefaultKeyLength(),
78     DPK + eax_sm4_dec.DefaultKeyLength(),
79     eax_sm4_dec.DefaultIVLength());
80 //protected_data.txt-待访问的文件
81 //验证并解密
82 // 将解密的结果存储于recover对象中
83 FileSource enc_file_Src("protected_data.txt", true,
84     new AuthenticatedDecryptionFilter(eax_sm4_dec,
85     new StringSink(recover)));
86 cout << "recover: " << recover << endl;

```

```

87     }
88     catch (const Exception& e)
89     { //出现异常
90         cout << e.what() << endl; //异常原因
91     }
92     return 0;
93 }

```

执行程序，并输入相同的口令（hanlulu1234567890cryptopp），运行结果如图2所示。

```

C:\windows\system32\cmd.exe
请输入访问数据的口令（不少于10个字符）：
hanlulu1234567890cryptopp
salt: 5E81694F5930DC2727AC4C46E7530AF218DCEDBFE6D29769EB9D40A98366C8778CE6B7E72A
25CAF07147016B2EB8BD60575C422533BBA3B0D0AB35186583B41C847719346C445022BEB4EC1920
CAD5579DDEF286899A2359D98B5868AA7731239E7B5CCBB86C011595F20C373578872BD5CC1736D
90435289FDD563026F7008
dpk_enc: F1AC6A7D818F487E4FAA335FE8DB24CC5C85F7C987C782F77733CB2637DE2F1B1AC16BB
4C2ABE6F63A2AB35EB005BA74
recover: Cryptography or cryptology is the practice and study of techniques for
secure communication in the presence of third parties called adversaries.
More generally, cryptography is about constructing and analyzing protocols that
prevent third parties or the public from reading private messages; various aspect
s in information security such as data confidentiality, data integrity, authenti
cation, and non-repudiation are central to modern cryptography.
Modern cryptography exists at the intersection of the disciplines of mathematics
, computer science, electrical engineering, communication science, and physics.

Applications of cryptography include electronic commerce, chip-based payment car
ds, digital currencies, computer passwords, and military communications.
请按任意键继续. . .

```

图2 数据访问程序运行结果

程序执行完毕后，recover对象中的内容即为我们要访问的明文数据，它是一段描述密码学的英文文字。这段文字与文件protecting_data.txt中的内容完全一致。

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》