OpenSSL 使用指南

0.6a

目录

0. 密码基础

本节介绍一些必须事先了解的密码学知识和密码算法。密码算法都是公开的，保密应该依赖于密钥的保密，而不是算法的保密。

0.1 对称加密

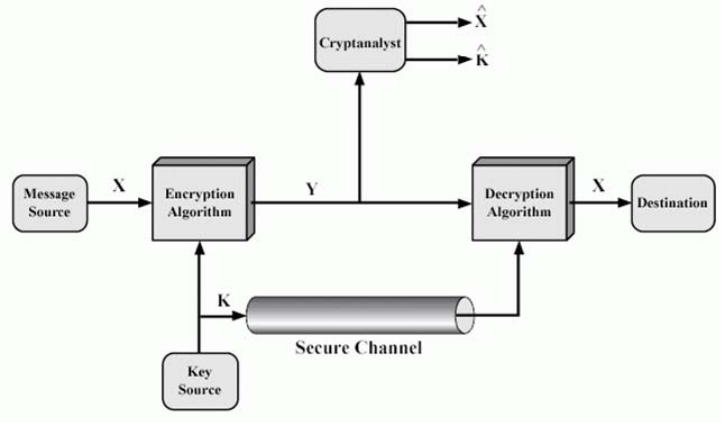
网络中分组和报文一般以明文方式传输。为了避免被偷听而泄露，需要加密。传统上使用对称算法加密，后来出现了非对称算法，即公钥算法。

对称加密算法中，明文 P，密钥 K，密文 C，加密算法 E，解密算法 D：

加密　E（P，K）= C

解密　D（C，K）= P

为了方便一般使 E=D。｛ 在理解上方便，可以认为 E=D=XOR。｝



主要问题在于达成一致密钥的困难性。由于密钥需要保密，因此需要事先秘密约定，或者用额外的安全信道来传输。

主要的对称算法有 DES、AES、RC4 等。

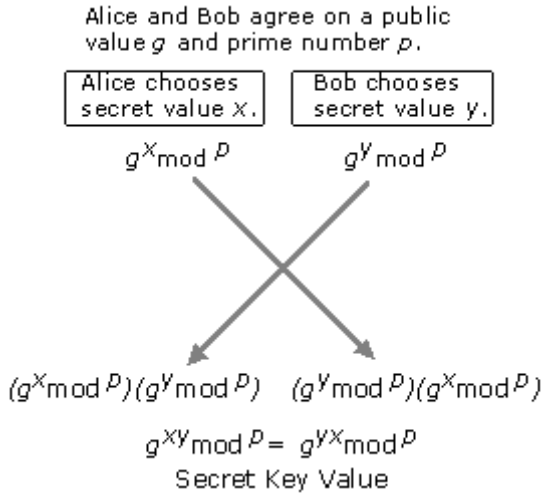
0.2 Diffie-Hellman 密钥协商协议

D-H 是一个用来在公开信道协商密钥的协议。D-H 协议依赖这样一个计算上困难的数学问题，即下式中求 y 是容易的，而知道 y 求 x 是困难的。该问题叫离散对数问题。

g^x mod p = y

A、B 双方各自选择秘密数 Xa 和 Xb，交换 Ya=g^Xa mod p 和 Yb=g^Xb mod p，并各自分别计算 Ka=Yb^Xa mod p 和 Kb=Ya^Xb mod p。事实上 Ka=Kb，而且，他人从 Y 得不到 K。

该协议简单有效，不依赖于任何事先的秘密约定，广泛用在 IPSec/VPN 等很多需要在线协商密钥的场合。但是该协议中没有关于通信双方身份的鉴别，因此存在中间人攻击。



0.3 公钥算法

密钥分为加密钥和解密钥两部分：K=(K1,K2)，K1 公开发布，且从 K1 难于推导出 K2。

加密　E（P，K1）= C

解密　D（C，K2）= P

下面介绍的 RSA 算法是公钥算法的典型代表和事实标准。

选取随机大素数 p 和 q，选定 e 计算 d 满足 ed=k(p-1)(q-1)+1，记 n=pq，则公钥(e,n)，私钥(d,n)。公钥要发布，私钥要保密。加密消息 m：

加密　m^e mod n = c

解密　c^d mod n = m

RSA 算法是基于因子分解问题的困难性的。另有基于离散对数问题的 ElGamal 类等公钥算法。

0.4 数字签名

RSA 算法可以用来签名。

用户对消息 m 的签名是通过用户对 m 执行了一种特定的操作，这种操作的结果能够被公开地验证，也能抗伪造。

对消息 m 的签名 s 定义为：

签名　m^d mod n = s

验证　s^e mod n =? m

0.5 HASH 算法

为了使用方便和提高速度，一般不是直接对 m 签名，而是对 m 的散列值签名。这要求找和 m 有相同散列值的消息 m’是困难的。

主要的 HASH 函数有 MD5 和 SHA1 等。

0.6 完整性

为了发现对消息的篡改，可以对消息做数字签名。

对于明文传输，也可以给明文增加带 key 的 HASH 校验值，这叫 MAC 码。

对于密文传输，也可以在加密前给明文增加不带 key 的 HASH 值，然后加密。

0.7 HMAC

用某种 Hash 算法结合 key 计算 MAC 的算法和标准。可以理解为

HMAC(Message, Key) := Hash(Message||Key)

发送时发送 Message||HMAC(Message, Key)，接收方考察是否还满足该结构特征，如果不满足就认为有破坏完整性的篡改企图发生。

与之相对，对密文的完整性保护是这样的：

ENC（Message||Hash(Message), Key）

0.9 混合密码体制

由于公钥算法太慢，因此实际使用的时候常用公钥算法传输会话密钥，再用对称算法和会话密钥加密批量数据。

签名也必须使用公钥算法。对称算法不能用来签名。

0.10 公钥的发布问题

公钥的发布是公钥体制的核心。一般设立权威中心，由中心以数字签名的方式签发公钥，即证书。证书中包含的信息主要有公钥、持有人信息、密钥用途说明、有效期、签发人信息和签发人的数字签名。

0.11 身份鉴别

如何鉴别通信的对方是否是所期望的人呢？

如果有了对方的证书/公钥，则可让对方签名来看看。如果签出能被这个公钥验证的名，则对方有对应私钥，是真的；否则就是假冒的。

1. 介绍

OpenSSL 是使用非常广泛的 SSL 的开源实现。由于其中实现了为 SSL 所用的各种加密算法，因此 OpenSSL 也是被广泛使用的加密函数库。

1.1 SSL

SSL(Secure Socket Layer)安全协议是由 Netscape 公司首先提出，最初用在保护 Navigator 浏览器和 Web 服务器之间的 HTTP 通信(即 HTTPS)。后来 SSL 协议成为传输层安全通信事实上的标准，并被 IETF 吸收改进为 TLS(Transport Layer Security)协议。

SSL/TLS 协议位于 TCP 协议和应用层协议之间，为传输双方提供认证、加密和完整性保护等安全服务。SSL 作为一个协议框架，通信双方可以选用合适的对称算法、公钥算法、MAC 算法等密码算法实现安全服务。

1.2 SSL 的工作原理

参见 http://developer.netscape.com/tech/security/ssl/howitworks.html 。

1.3 OpenSSL

OpenSSL 是著名的 SSL 的开源实现，是用 C 语言实现的。

OpenSSL 的前身是 SSLeay，一个由 Eric Young 开发的 SSL 的开源实现，支持 SSLv2/v3 和 TLSv1。

伴随着 SSL 协议的普及应用，OpenSSL 被广泛应用在基于 TCP/Socket 的网络程序中，尤其是 OpenSSL 和 Apache 相结合，是很多电子商务网站服务器的典型配置。

2. 编译和安装 OpenSSL

OpenSSL 开放源代码，这对学习、分析 SSL 和各种密码算法提供了机会，也便于在上面进一步开发。

2.1 获得 OpenSSL

到 OpenSSL 的网站即可下载当前版本的 OpenSSL 源代码压缩包。

当前版本 openssl-0.9.8.tar.gz，只有 3M 多，比较精简。解压缩后得到一个目录 openssl-0.9.8，共有约 1800 个文件，15M。其中 crypto 子目录中是众多密码算法实现，ssl 子目录中是 SSL 协议的实现。

在 Linux 中解压缩：

```
$tar zxf openssl-0.9.8.tar.gz
```

在 Windows 中可以使用 winzip 或 winrar。

2.2 编译工具

编译 OpenSSL 需要 Perl 和 C 编译器。在 Windows 下如果要用加密算法的汇编代码实现，还需要 masm 或 nasm 汇编器。(汇编代码可以比 C 代码显著提高密码运算速度)

Perl 在 Windows 下推荐使用 Active Perl。

C 编译器可以使用 gcc。在 Windows 下可以使用 Visual C++ 编译器。

汇编器推荐使用 nasm。

这些工具所在目录必须加入到 PATH 环境变量中去。

2.3 编译和安装步骤

查看./readme 是个好习惯。从 readme 了解到需要进一步查看 INSTALL 和 INSTALL.W32 文件。

在 Windows 中：

```
>perl Configure VC-WIN32
>ms\do_nasm (如果不使用汇编代码实现，则可>ms\do_ms)
>nmake -f ms\ntdll.mak
>cd out32dll
>..\ms\test
```

编译结果得到头文件、链接库、运行库和 openssl.exe 工具。头文件位于./inc32 或者./inculde 目录，有一个 openssl 子目录，内有几十个.h 文件。链接库即./out32dll 目录中的 libeay32.lib 和 ssleay32.lib，分别是密码算法相关的和 ssl 协议相关的。运行库是./out32dll 目录中的 libeay32.dll 和 ssleay32.dll，和链接库相对应。在./out32dll 中还有一个工具 openssl.exe，可以直接用来测试性能、产生 RSA 密钥、加解密文件，甚至可以用来维护一个测试用的 CA。

在 Linux 中的编译和安装步骤较简单：

```
$./config
$make
$make test
$make install
```

在 Linux 下，头文件、库文件、工具都已被安装放到了合适的位置。库文件是.a 或.so 格式。

3. 使用 OpenSSL.exe

使用 OpenSSL.exe(Linux 中可执行文件名是 openssl)可以做很多工作，是一个很好的测试或调试工具。

3.1 版本和编译参数

显示版本和编译参数： >openssl version -a

3.2 支持的子命令、密码算法

查看支持的子命令： >openssl ?

SSL 密码组合列表： >openssl ciphers

3.3 测试密码算法速度

测试所有算法速度： >openssl speed

测试 RSA 速度： >openssl speed rsa

测试 DES 速度： >openssl speed des

3.4 RSA 密钥操作

产生 RSA 密钥对： >openssl genrsa -out 1.key 1024

取出 RSA 公钥： >openssl rsa -in 1.key -pubout -out 1.pubkey

3.5 加密文件

加密文件： >openssl enc -e -rc4 -in 1.key -out 1.key.enc

解密文件： >openssl enc -d -rc4 -in 1.key.enc -out 1.key.dec

3.6 计算 Hash 值

计算文件的 MD5 值： >openssl md5 < 1.key

计算文件的 SHA1 值： >openssl sha1 < 1.key

4. 算法编程 API

OpenSSL 中支持众多的密码算法，并提供了很好的封装和接口。密码算法主要分为如下几类：对称算法、公钥算法、散列算法、随机数产生算法等。

OpenSSL 的目标是实现安全协议。其中相关协议和标准包括：SSL/TLS、PKCS#1、PCKS#10、X.509、PEM、OCSP 等。

4.1 对称算法接口

OpenSSL 中实现的对称算法太多，举三个例子：DES、AES、RC4。

4.1.1 DES

DES 加密算法是分组算法。DES 的基本操作是把 64 比特明文在 56 比特密钥指引下加密成 64 比特密文。在实际使用中把密钥看作 64 比特可以更方便。

DES（IN, KEY）= OUT

(1) DES ECB 模式

在 OpenSSL 中 ECB 操作模式对应的函数是 DES_ecb_encrypt()，该函数把一个 8 字节明文分组 input 加密成为一个 8 字节密文分组 output。参数中密钥结构 ks 是用函数 DES_set_key()准备好的，而密钥 key 是用随机数算法产生的 64 个随机比特。参数 enc 指示是加密还是解密。该函数每次只加密一个分组，因此用来加密很多数据时不方便使用。

```
void DES_ecb_encrypt(const_DES_cblock *input,DES_cblock *output,
DES_key_schedule *ks,int enc);
int DES_set_key(const_DES_cblock *key,DES_key_schedule *schedule);
```

(2) DES CBC 模式

DES 算法 CBC 操作模式加解密函数是 DES_ncbc_encrypt()。参数 length 指示输入字节长度。如果长度不是 8 字节的倍数，则会被用 0 填充到 8 字节倍数。因此，输出可能比 length 长，而且必然是 8 字节的倍数。

```
void DES_ncbc_encrypt(const unsigned char *input,unsigned char *output, long
length, DES_key_schedule *schedule, DES_cblock *ivec, int enc);
```

(3) DES CFB 模式

DES 算法 CFB 操作模式加解密函数是 DES_cfb_encrypt()。参数 length 指示输入字节长度。参数 numbits 则指示了 CFB 每次循环加密多少明文比特，也即密文反馈的比特数目。ivec 是初始向量，被看做第 0 个密文分组，是不用保密但应随机取值的 8 个字节。如果在一次会话中数次调用 DES_cfb_encrypt()，则应该记忆 ivec。由于 CFB 模式中每次 DES 基本操作只加密 numbits 比特明文，因此如果 numbits 太小则效率太低。

```
void DES_cfb_encrypt(const unsigned char *in, unsigned char *out, int numbits,
long length, DES_key_schedule *schedule, DES_cblock *ivec, int enc);
```

另有一个 numbit 是 64 比特的版本，既高效又没有填充的麻烦，推荐使用。num 中的返

回值指示了 ivec 中的状态，是和下次调用衔接的。

```
void DES_cfb64_encrypt(const unsigned char *in, unsigned char *out, long length,
DES_key_schedule *schedule, DES_cblock *ivec, int *num, int enc);
```

(4) DES OFB 模式

OFB 和 CFB 类似，也有两个函数，用法一样。

```
void DES_ofb_encrypt(const unsigned char *in,unsigned char *out,int
numbits,long length,DES_key_schedule *schedule,DES_cblock *ivec);
void DES_ofb64_encrypt(const unsigned char *in,unsigned char *out,long
length,DES_key_schedule *schedule,DES_cblock *ivec,int *num);
```

(5) DES 函数示例程序

见附件 A.1。

4.1.2 AES

AES 加密算法是分组算法。典型参数的 AES 的基本操作是把 128 比特明文在 128 比特密钥指引下加密成 128 比特密文。

AES（IN，KEY）= OUT

OpenSSL 中关于 AES 的函数名和参数接口和 DES 的雷同。相关函数名如下(参数略)。

```
int AES_set_encrypt_key();
int AES_set_decrypt_key();
void AES_ecb_encrypt();
void AES_cbc_encrypt();
void AES_cfb128_encrypt();
void AES_ofb128_encrypt();
```

AES 示例程序见附件 A.2。

4.1.3 RC4

RC4 密码算法是流算法，也叫序列算法。流算法是从密钥作为种子产生密钥流，明文比特流和密钥流异或即加密。RC4 算法由于算法简洁，速度极快，密钥长度可变，而且也没有填充的麻烦，因此在很多场合值得大力推荐。

OpenSSL 中 RC4 算法有两个函数：RC4_set_key()设置密钥，RC4()加解密。可以把 RC4 看作异或，因此加密两次即解密。

```
void RC4_set_key(RC4_KEY *key, int len, const unsigned char *data);
void RC4(RC4_KEY *key, unsigned long len, const unsigned char *indata, unsigned
char *outdata);
```

RC4 示例程序见附件 A.3。

例子 A.3.(1)是利用 OpenSSL 动态库函数。例子 A.3.(2)是把 RC4 的实现代码从 OpenSSL 中分离出来的。例子 A.3.(3)是另一个演示实现。

4.2 公钥算法

OpenSSL 中实现了 RSA、DSA、ECDSA 等公钥算法。

4.2.1 RSA

RSA 是分组算法，典型的密钥模长度 1024 比特时，分组即是 1024 比特，即 128 字节。

(1) RSA 密钥

RSA 密钥产生函数 RSA_generate_key()，需要指定模长比特数 bits 和公钥指数 e。另外两个参数为 NULL 即可。

```
RSA * RSA_generate_key(int bits, unsigned long e, void (*callback)(int,int,void *),void *cb_arg);
```

如果从文件中读取密钥，可使用函数 PEM_read_bio_PrivateKey()/PEM_read_bio_PUBKEY();EVP_PKEY 中包含一个 RSA 结构，可以引用。

```
EVP_PKEY *PEM_read_bio_PrivateKey(BIO *bp, EVP_PKEY **x, pem_password_cb *cb, void *u);
```

(2) RSA 加密解密

RSA 加密函数 RSA_public_encrypt()使用公钥部分，解密函数 RSA_private_decrypt()使用私钥。填充方式常用的有两种 RSA_PKCS1_PADDING 和 RSA_PKCS1_OAEP_PADDING。出错时返回-1。输入必须比 RSA 钥模长短至少 11 个字节（在 RSA_PKCS1_PADDING 时？）。输出长度等于 RSA 钥的模长。

```
int RSA_public_encrypt(int flen, const unsigned char *from,unsigned char *to, RSA *rsa,int padding);
int RSA_private_decrypt(int flen, const unsigned char *from,unsigned char *to, RSA *rsa,int padding);
```

(3) 签名和验证

签名使用私钥，验证使用公钥。RSA 签名是把被签署消息的散列值编码后用私钥加密，因此函数中参数 type 用来指示散列函数的类型，一般是 NID_md5 或 NID_sha1。正确情况下返回 0。

```
int RSA_sign(int type, const unsigned char *m, unsigned int m_length, unsigned char *sigret, unsigned int *siglen, RSA *rsa);
int RSA_verify(int type, const unsigned char *m, unsigned int m_length, unsigned char *sigbuf, unsigned int siglen, RSA *rsa);
```

(4) RSA 函数示例程序

RSA 示例程序见附件 A.4。

例子 A.4.(1)是加密解密例子。例子 A.4.(2)是签名验证例子。

4.2.2 DSA

（TOBE）

4.2.2 ECDSA

（or NOT TOBE）

4.3 Hash 算法

Hash 算法举 MD5 和 SHA1 两个例子。Hash 算法重复接收用户输入，直到最后一次结束时输出散列结果。

4.3.1 MD5

MD5 算法输出的散列值是 16 字节。

```
int MD5_Init(MD5_CTX *c);
int MD5_Update(MD5_CTX *c, const void *data, size_t len);
int MD5_Final(unsigned char *md, MD5_CTX *c);
```

4.3.2 SHA1

SHA1 算法输出的散列值是 20 字节。

```
int SHA1_Init(SHA_CTX *c);
int SHA1_Update(SHA_CTX *c, const void *data, size_t len);
int SHA1_Final(unsigned char *md, SHA_CTX *c);
```

4.3.3 MD5 例子

MD5 示例程序见附件 A.5。

md5sum 这是一个实用小工具，可以计算一个文件的 MD5 值。

4.4 随机数算法

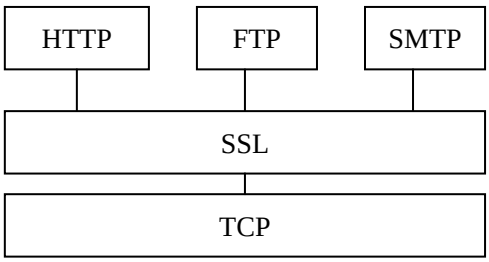
随机性是密码安全的基石。为了产生安全的伪随机数，必须有好的随机因素作为种子。OpenSSL 在内部做了努力，但是仍建议在实用随机数产生函数之前添加随机因素。

函数 RAND_add()可以添加随机因素到内部状态中去。然后，即可以使用 RAND_bytes()获得随机数。

```
void RAND_add(const void *buf,int num,double entropy);
int  RAND_bytes(unsigned char *buf,int num);
```

5. SSL 协议编程 API

SSL 协议的主要功能即把 TCP 的字节流变成了一个安全的流，所有基于 TCP 的程序可以很容易地采用 SSL 协议。主要的变化在于 accept/connect()变化为使用 OpenSSL 提供的 SSL_accept/SSL_connect()，read/write()变化为 SSL_read/SSL_write()。



5.0 初始化函数库

```
SSL_load_error_strings();
OpenSSL_add_all_algorithms();
```

5.1 客户端

```
SSLv3_client_method();
SSL_CTX_new();
SSL_CTX_set_accepted_Cas();     // 认可的 CAs
SSL_CTX_use_certificate_file(); // 自己的证书
SSL_CTX_use_PrivateKey_file();  // 自己的私钥
SSL_CTX_check_private_key();    // 检查证书-私钥一致性
SSL_CTX_set_cipher();           // 自己喜欢的算法组合
```

5.2 服务器端

```
SSLv3_server_method();
SSL_CTX_new();
SSL_CTX_set_accepted_Cas();
SSL_CTX_use_certificate_file();
SSL_CTX_use_PrivateKey_file();
SSL_CTX_check_private_key();
SSL_CTX_set_cipher();
```

5.3 SSL 示例程序

参见 A.6。

6. CA 和证书

公钥是不需要保密的。事实上，公钥需要可靠地发布、彻底地公开。很不幸，公钥的公开是应用密码学的最后一个难题：当你面对一个公钥（就是两个数 n 和 e）时，你怎么知道这是谁的公钥？

为了让人相信这个公钥就是某个人的，一般要有一个权威机构(即 CA)的保证。CA 用自己的私钥给公钥和持有人信息签名。任何人可以用 CA 的公钥验证签名，以此相信这个公钥是这个人的。

CA 的公钥怎么来呢？一般是事先内置在程序中的，或者通过其他可靠的方法取得。

6.0 CA 的作用

OpenSSL 中有一个小巧但可以用作测试的 CA。如果想使用一个功能全面强大的 CA，建议使用 EJBCA 或 OpenCA。Windows Server 中有个可选的 CA 组件，对于支持 IIS、IE 和 Outlook 中的使用是方便的。使用商业 CA（如山东省 CA）颁发的证书应该也是可以的。

6.1 OpenSSL 中 CA 的配置示例

参见 A.7.(1)。

6.3 证书解析和示例程序

参见 A.7.(2)。

7. 参考网址

SSL 3.0 Specification	http://www.netscape.com/eng/ssl3/
Transport Layer Security (tls) Charter	http://www.ietf.org/html.charters/tls-charter.html
OpenSSL: The Open Source toolkit for SSL/TLS	http://www.openssl.org/
SSLeay	http://www2.psy.uq.edu.au/~ftp/Crypto/
OpenSSL中文论坛	http://openssl.cn/
Perl	http://www.cpan.org/src/README.html
ActivePerl	http://www.activestate.com/Products/ActivePerl/
NASM	http://www.perl.com/
山东省CA	http://www.sdca.com.cn/

8. TO BE

本文档为“应用密码学和信息安全”课程讲义的配套文档。

更新发布在http://bellflower.3322.org/ 。

如有疑问，请联系linfb@sdu.edu.cn 。

A. 示例程序

注:此嵌入的文件对象可以被“拖放”到磁盘目录中去。

1. DES 示例程序

Demo_des.zip

2. AES 示例程序

Demo_aes.zip

3. RC4 示例程序

enc.c

(1).

Rc4.zip

(2).

Myrc4.zip

(3)

4. RSA 示例程序

demo_rsaenc,v2.zip

(1).

demo_sign,v2.zip

(2).

5. Hash 算法示例程序

md5sum.zip

6. SSL 示例程序

demo_mini.zip

7. CA 配置示例和证书解析示例程序

oCA-config.zip

(1).

demo_cert.zip

(2).

8.

B.