**第** **1** **2** **章**

**SSL-TLS编程**

**12.1 SSL** **协议规范**

**12.1.1什么是SSL** **协议**

安全套接字层(Secure Sockets Layer,SSL)协议是一个中间层协议，它位于TCP/IP 层和 应用层之间，为应用层程序提供一条安全的网络传输通道。它的主要目标是在两个通信应用之 间提供私有性和可靠性。SSL协议由两层组成，最低层是SSL 记录层协议(SSL Record Protocol), 它基于可靠的传输层协议(如TCP), 用于封装各种高层协议；高层协议主要包括SSL 握手 协议(SSL Handshake Protocol)、改变加密规约协议(Change Cipher Spec Protocol)、告警协 议 (Alert Protocol) 等。

**12.1.2** **SSL协** **议** **的** **优** **点**

SSL 协议的一个优点是它与应用层协议无关，一个高层的协议可以透明地位于 SSL 协议 层的上方。SSL 协议提供的安全连接具有以下几个基本特性：

(1)连接是安全的，在初始化握手结束后，SSL 使用加密方法来协商一个秘密的密钥， 数据加密使用对称密钥技术(如DES 、RC4等 ) 。

(2)可以通过非对称(公钥)加密技术(如RSA 、DSA) 等认证对方的身份。

(3)连接是可靠的，传输的数据包含数据完整性的校验码，使用安全的哈希函数(如SHA、 MD5等)计算校验码。

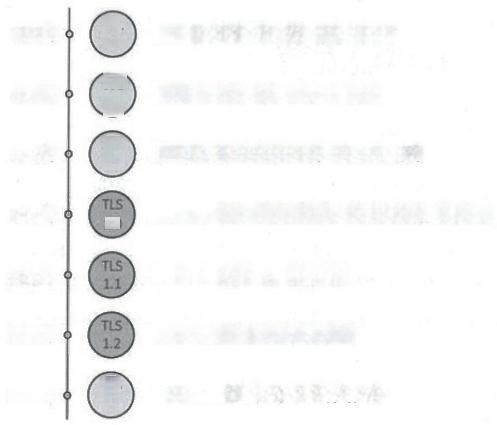
**12.1.3** **SSL协议的发展**

SSL v1.0最早由网景公司(NetScape, 以浏览器闻名)在1994年提出，该方案第一次解 决了安全传输的问题。1995 年公开发布了 SSL v2.0,该方案于2011 年被弃用 (RFC6176-Prohibiting Secure Sockets Layer(SSL)Version 2.0) 。1996年发布了SSL v3.0(2011 年才补充的RFC 文档：RFC 6101-The Secure Sockets Layer(SSL)Protocol Version 3.0),被 大规模应用，于2015年弃用(RFC7568-Deprecating Secure Sockets Layer(SSL)Version 3.0)。



Windows C/C++加密解密实战

这之后经过几年发展，于1999年被IETF 纳入标准化(RFC2246-The TLS Protocol Version 1.0), 改名叫TLS(Transport Layer Security Protocol,安全传输层协议),和SSL v3.0相比几乎没有 什么改动。2006年提出了TLS v1.1(RFC4346-The Transport Layer Security(TLS)Protoco Version 1.1),修复了一些Bug,支持更多参数。2008年提出了TLS v1.2(RFC5246-The Transport Layer Security(TLS)Protocol Version 1.2),做了更多的扩展和算法改进，是目前几乎所有新 设备的标配。TLS v1.3在2014年已经提出，2016年开始草案制定，然而由于TLS v1.2 的广 泛应用，必须考虑到支持v1.2 的网络设备能够兼容 v1.3, 因此反复修改直到第28个草案才于 2018年正式纳入标准(The Transport LayerSecurity Protocol Version 1.3) 。TLSv1.3改善了握 手流程，减少了时延，并采用安全的密钥交换算法。图12-1演示了SSL 的发展。



|  |  |  |
| --- | --- | --- |
| 1994年 | SSL 1.0 | NetScape公司提出SSL第一版，未公开 |
| 1995年 | SSL  2.0 | 公开发布了第二版，于2011年弃用 |
| 1996年 | SSL 3.0 | 第三版得到了大规模应用，于2015年弃用 |
| 1999年  2006年 | 1.0 | RFC2246:被IETF纳入标准化，没太大改动，改名TLS  RFC4346:修复Bug,增加参数 |
| 2008年 |  | RFC5246:更多扩展和算法改进 |
| 2018年 | TLS 1.3 | RFC8446:减少时延，完全前向安全 |

图12-1

12.1.4 SSL v3/TLS提 供 的 服 务

(1)客户方和服务器的合法性认证

保证通信双方能够确信数据将被送到正确的客户方或服务器上。客户方和服务器都有各自 的证书。为了验证用户，SSL/TLS 要求双方在交换证书以进行身份认证的同时获取对方的公 钥。

(2)对数据进行加密

使用的加密技术既有对称算法，又有非对称算法。具体地说，在安全的连接建立起来之前， 双方先用非对称算法加密握手信息和进行对称算法密钥交换，安全连接建立之后，双方用对称 算法加密数据。

(3)保证数据的完整性

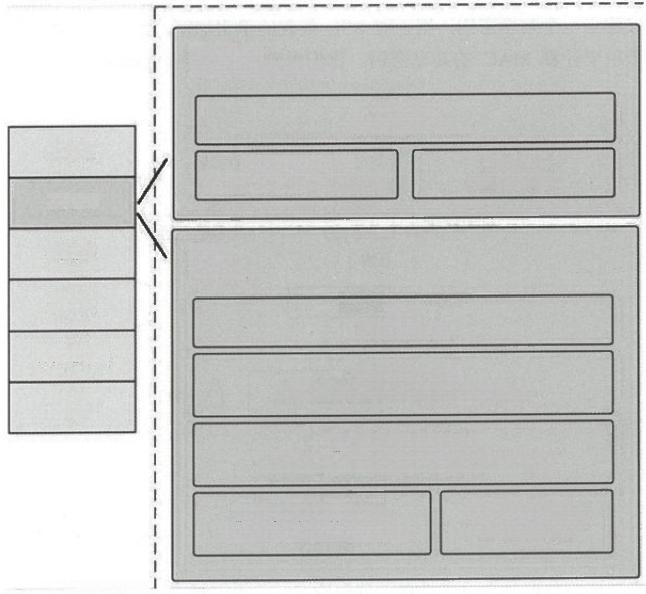
采用消息摘要函数(MAC) 提供数据完整性服务。



**第12章** **SSL-TLS** **编程**

**12.1.5** **SSL协议层次结构模型**

SSL 协议是一个分层的协议，由两层组成。SSL 协议的层次结构如图12-2所示。



握手协议层

Handshake Protocol Layer

握手协议

Handshake Protoxol

应用层

更换加密规约协议

Change Gper Sepc Protocol

SSL v3.0

记录层

Record Layer

分片

Fragmentation

数据链路层

记录压缩和解压缩

Record Compression and Decompression

物理层

记录负载保护和加密规约

Record PayloadProtection and the Cpher Spec

空值或者标准流加密算法 加密块链块加密算法

Null or Standard Stream Cipher CBC Block Cipher

传输层

网络层

告警协议

Alert Protocol

图12-2

SSL 记录协议(SSL Record Protocol): 建立在可靠的传输协议(如TCP) 之上，为高层 协议提供数据封装、压缩、加密等基本功能的支持。

SSL 握手协议(SSL Handshake Protocol): 建立在 SSL 记录协议之上，用于在实际的数 据传输开始前，通信双方进行身份认证、协商加密算法、交换加密密钥等。SSL 协议实际上是 SSL 握手协议、SSL 修改密文协议、SSL 警告协议和SSL 记录协议组成的一个协议族。SSL 握手协议是SSL 协议的核心。

**12.1.6 SSL记录层协议**

SSLv3/TLS 记录层协议是一个分层的协议。每一层都包含长度、描述和数据内容。记录 层协议把要传送的数据、消息进行分段，可能还会进行压缩，最后进行加密传送。对输入数据 解密、解压、校验，然后传送给上层调用者。

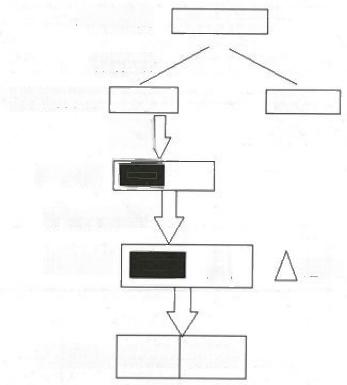
协议中定义了4种记录层协议的调用者：握手协议、告警协议、加密修改协议、应用程序 数据协议。为了允许对协议进行扩展，对其他记录类型也可以支持。任何新类型都必须另外分 配其他的类型标志。如果一个 SSLv3/TLS 要实现接收它不能识别的记录类型，就必须将其丢



**Windows C/C++加密解密实战**

弃。运行于 SSLv3/TLS之上的协议必须注意防范基于这点的攻击。因为长度和类型字段是不 受加密保护的，所以必须小心非法用户可能针对这一点进行使用分析。

SSL 记录协议可为SSL 连接提供保密性业务和消息完整性业务。保密性业务是通信双方 通过握手协议建立一个共享密钥，用于对 SSL 负载的单钥加密消息。完整性业务是通过握手 协议建立一个用于计算MAC 的共享密钥。我们来看一个记录层协议的执行过程，如图12-3 所示。



应用层数据 abcdefghi

分段组合

记录协议单元 abcd fghi

压缩

压缩单元

加密并计算MAC

加密数据 △ 是 MAC

TCP 数 据 包 TCP 报头 数据体

图12-3

SSL 将被发送的数据分为可供处理的数据段(这个过程称为分片或分段),它没有必要去 解释这些数据，并且这些数据可以是任意长度的非空数据块。接着对这些数据进行压缩、加密， 然后把密文交给下一层网络传输协议处理。对于收到的数据，处理过程与上面相反，即解密、 验证、解压缩、拼装，然后发送到更高层的用户。

**1.分片**

SSL 记录层把上层送来的数据块切分成以16KB为单位的SSL 明文记录块，最后一块可 能不足16KB。在记录层中，并不保留上层协议的消息边界，也就是说，同一内容类型的多个 上层消息可以被连接起来，封装在同一SSL 明文记录块中。不同类型的消息内容还是会被分 离处理，应用层数据的传输优先级一般比其他类型的优先级低。

**2.记录块的压缩和解压缩**

被切分后的记录块将使用当前会话状态中定义的压缩算法来压缩。一般来说，都会有一个 压缩算法被激活，但在初始化时都被设置成使用空算法(不使用数据压缩)。压缩算法将SSL 明文记录转化为SSL 压缩记录。使用的压缩必须是无损压缩，而且不能使压缩后的数据长度 增加超过1024B(在原来的数据就已经是压缩数据时，再使用压缩算法就可能因添加了压缩信 息而增大)。



**第12章** **SSL-TLS** **编程**

**3.记录负载的保护**

所有的记录都会用当前的密码约定中定义的加密算法和 MAC 算法来保护。通常都会有一 个激活的加密约定，但是在初始化时，加密约定被定义为空，这意味着并不提供任何的安全保 护。

一旦握手成功，通信双方就共享一个会话密钥，这个会话密钥用来加密记录，并计算它们 的消息校验码 (MAC) 。 加密算法和MAC函数把SSL 压缩记录转换成SSL 密文记录；解密 算法则进行反向处理。

**12.1.7 SSL握手协议层**

**1.握手协议**

握手协议在SSL 记录层之上，它产生会话状态的密码参数。当SSL 客户端和服务器开始 通信时，它们协商一个协议版本，选择密码算法对彼此进行验证，使用公开密钥加密技术产生 共享密钥。这些过程在握手协议中进行。

SSL 协议既用到了公钥加密技术(非对称加密),又用到了对称加密技术，SSL 对传输 内容的加密采用的是对称加密，然后对对称加密的密钥使用公钥进行非对称加密。这样做的好 处是，对称加密技术比公钥加密技术速度快，可用来加密较大的传输内容，公钥加密技术相对 较慢，提供了更好的身份认证技术，可用来加密对称加密过程使用的密钥。

SSL 的握手协议非常有效地让客户和服务器之间完成相互之间的身份认证，其主要过程如下：

(1)客户端的浏览器向服务器传送客户端 SSL 协议的版本号、加密算法的种类、产生 的随机数以及其他服务器和客户端之间通信所需要的各种信息。

(2)服务器向客户端传送 SSL 协议的版本号、加密算法的种类、随机数以及其他相关 信息，同时服务器还将向客户端传送自己的证书。

(3)客户端利用服务器传过来的信息验证服务器的合法性，服务器的合法性包括：证书 是否过期、发行服务器证书的 CA 是否可靠、发行者证书的公钥能否正确解开服务器证书的 “发行者的数字签名”、服务器证书上的域名是否和服务器的实际域名相匹配。如果合法性验 证没有通过，通信将中断；如果合法性验证通过，将继续进行第(4)步。

(4)客户端随机产生一个用于后面通信的“对称密码”,用服务器的公钥(服务器的公钥 从步骤(2)中的服务器的证书中获得)对其加密，然后将加密后的“预主密码”传给服务器。

(5)如果服务器要求客户的身份认证(在握手过程中为可选),用户可以建立一个随机 数，然后对其进行数据签名，将这个含有签名的随机数和客户自己的证书以及加密过的“预主 密码”一起传给服务器。

(6)如果服务器要求客户的身份认证，服务器必须检验客户证书和签名随机数的合法性， 具体的合法性验证过程包括：客户的证书使用日期是否有效、为客户提供证书的CA 是否可 靠、发行CA 的公钥能否正确解开客户证书的发行 CA 的数字签名、检查客户的证书是否在 证书废止列表(CRL) 中。检验如果没有通过，通信立刻中断；如果验证通过，服务器将用自



**Windows C/C++加密解密实战**

己的私钥解开加密的“预主密码”,然后执行一系列步骤来产生主通信密码(客户端也将通 过同样的方法产生相同的主通信密码)。

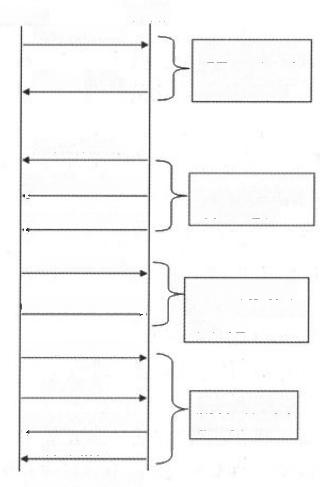
(7)服务器和客户端用相同的主密码，即“通话密码”,一个对称密钥用于 SSL 协议 的安全数据通信的加解密通信。同时，在 SSL 通信过程中还要完成数据通信的完整性，防止 数据通信中的任何变化。

(8)客户端向服务器端发出信息，指明后面的数据通信将使用步骤(7)中的主密码为对 称密钥，同时通知服务器客户端的握手过程结束。

(9)服务器向客户端发出信息，指明后面的数据通信将使用步骤(7)中的主密码为对称 密钥，同时通知客户端服务器的握手过程结束。

(10)SSL 的握手部分结束，SSL 安全通道的数据通信开始，客户端和服务器开始使用 相同的对称密钥进行数据通信，同时进行通信完整性的检验。

简而言之，握手过程可以用图12-4来表示。



客户端 服务器 Client Hello

Server Hello

Certificate

Cortificate Requost

ServerHello Done

Certificate

Certiflcate Verlfy

ChangeClpherSpec

Finished

ÇhangeCipherSpec Finished

Applicatlon Data

如果服务器需要， 则发送客户的证书 作为响应

改变加密算法约 定，握手结束

建立协议版本号、 会话ID、密码组 件、压编方法

发送服务器证书和 请求客户证书

图12-4

在客户端发送Client Hello信息后，对应的服务器回应Server Hello信息，否则产生一个致 命错误，导致连接失败。Client Hello和 Server Hello用于在客户端和服务器之间建立安全增强 功能，并建立协议版本号、会话标识符、密码组和压缩方法。此外，产生和交换两组随机值：

ClientHello.random 和 ServerHello.random。

在Hello信息之后，如果需要被确认，服务器将发送其证书信息。如果服务器被确认，并 且适合所选择的密码组，就需要对客户端请求证书信息。

现在，服务器将发送Server Hello Done信息，表示握手阶段的 Hello 信息部分已经完成， 服务器将等待客户端响应。



**第** **1** **2** **章** **SSL-TLS** **编程**

如果服务器已发送了一个证书请求(Certificate Request)信息，客户端可回应证书信息或 无证书(No Certificate)警告。然后发送Client Key Exchange信息，信息的内容取决于在Client Hello 和 Server Hello之间选定的公开密钥算法。如果客户端发送一个带有签名能力的证书， 服务器发送一个数字签名的Certificate Verify信息用于检验这个证书。

这时，客户端发送一个ChangeCipherSpec 信息，将PendingCipherSpec(待用密码参数) 复制到CurrentCipherSpec(当前密码参数),然后客户端立即在新的算法、密钥和密码下发送 结 束(Finished) 信息。对应地，如果服务器发送自己的ChangeCipherSpec信息，并将 Pending Cipher Spec复制到Current Cipher Spec,然后在新的算法、密钥和密码下发送结束信息。这一 时刻，握手结束。客户端和服务器可开始交换其应用层数据。

下面介绍HandshakeType的各类信息。

(1)Hello Request(问候请求)

服务器可在任何时候发送信息，如果客户端正在一次会话中或者不想重新开始会话，客户 端可以忽略这条信息。如果服务器没有和客户端进行会话，发送了Hello Request,而客户端没 有发送Client Hello,就会发生致命错误，关闭同客户端的连接。

(2)Client Hello(客户端问候)

当客户端第一次连接到服务器时，应将 Client Hello作为第一条信息发给服务器。Client Hello包含客户端支持的所有压缩算法，如果服务器均不支持，则本次会话失败。

(3)Server Hello(服务器问候)

Server Hello信息的结构类似于Client Hello,它是服务器对客户端的ClientHello信息的回复。

(4)Server Certificate(服务器证书)

如果要求验证服务器，则服务器立刻在 Server Hello信息后发送其证书。证书的类型必须 适合密钥交换算法，通常为x.509 v3证书或改进的x.509 证书。

(5)Certificate Request(证书请求)

如果和所选密码组相适应，服务器可以向客户端请求一个证书。如果服务器是匿名的，则 在请求客户端证书时会导致致命错误。

(6)Server Hello Done(服务器问候结束)

服务器发出该信息表明Server Hello结束，然后等待客户端响应。客户端收到该信息后检 查服务器提供的证书是否有效，以及服务器的 Hello参数是否可接受。

(7)Client Certificate(客户端证书)

该信息是客户端收到服务器的Server Hello Done后可以发送的第一条信息。只有当服务器 请求证书时才需发此信息。如果客户端没有合适的证书，则发送“没有证书”的警告信息，如 果服务器要求有“客户端验证”,则收到警告后宣布握手失败。

(8)Client KeyExchange(客户端密钥交换) 信息的选择取决于采用哪种公开密钥算法。



Windows C/C++加密解密实战

(9)Certificate Verify(证书检查)

该信息用于提供客户端证书的验证。它仅在具有签名能力的客户端证书之后发送。

(10)Finished (结束)

该信息在 Change Cipher Spec 之后发送，以证明密钥交换和验证的过程已顺利进行。发方 在发出 Finished 信息后可立即开始传送秘密数据，接收方在收到Finished 信息后必须检查其 内容是否正确。

**2.更换加密规约协议**

更换加密规约协议的存在是为了使密码策略能得到及时的通知。该协议只有一个消息(是 一个字节的数值),传输过程中使用当前的加密约定来加密和压缩，而不是改变后的加密约定。

客户端和服务器都会发出改变加密约定的消息，通知接收方后面发送的记录将使用刚刚协 商的加密约定来保护加密约定的消息；服务器则在成功处理从客户端接收到的密钥交换消息后 发送。一个意外改变的加密约定消息将导致一个Unexpected Message告警。当恢复之前的会 话时，改变加密约定消息将在问候消息后发送。

**3.告警协议**

告警协议是SSL 记录层支持的协议之一。告警消息传送该消息的严重程度和该告警的描 述。告警消息的致命程度会导致连接立即终止。在这种情况下，同一会话的其他连接可能还将 继续，但必须使会话的标识符失效，以防止失败的会话还继续建立新的连接。与其他消息一样， 告警消息也经过加密和压缩，使用当前连接状态的约定。

(1)关闭告警

为了防止截断攻击 (Truncation Attack), 客户端和服务器必须都知道连接已经结束了。 任何一方都可以发起关闭连接，发送Close Notify 告警消息，在关闭告警之后收到的数据都会 被忽略。

(2)错误告警

SSL 握手协议中的错误处理很简单：当检测到错误时，检测的这一方就发送一个消息给另 一方，传输或接收到一个致命告警消息，双方都马上关闭连接，要求服务器和客户端都清除会 话标识、密钥以及与失败连接有关的秘密。错误告警包括：意外消息告警、记录MAC 错误告 警、解压失败告警、握手失败告警、缺少证书告警、已破坏证书告警、不支持格式证书告警、 证书已作废告警、证书失效告警、不明证书发行者告警以及非法参数告警。

**12.2 OpenSSL** **中的** **SSL** **编程**

在了解了 SSL 协议的基本原理后，我们就可以进入实战环节了。OpenSSL 实现了SSL 协 议1.0、2.0、3.0以及TLS协议1.0。我们可以利用 OpenSSL提供的函数进行安全编程，这些



**第** **1** **2** **章** **SSL-TLS 编程**

函数定义在 openssl/ssl.h 文件中。

我们利用 SSL 编程主要是开发安全的网络程序。网络编程常见的套路是套接字编程，而 基 于OpenSSL 进行SSL 编程就是安全的套接字编程，其过程和普通的套接字编程类似。

OpenSSL 中提供和普通 Socket类似的函数，如常用的connect、acep、write、read,对 应 OpenSSL 中的 SSL\_connect 、SSL\_accept 、SSL\_wrie 、SSL\_read。不同的是OpenSSL 还需要设 置其他环境参数，如服务器证书等。

**12.3 SSL函数**

**12.3.1** **初始化** **SSL** **算法库函数** **SSL** **library\_init**

该函数用于初始化SSL 算法库，在调用SSL 系列函数之前必须先调用此函数。函数声明 如下：

int SSL\_library\_init();

若函数执行成功则返回1,否则返回0。 也可以用下列两个宏定义：



#define OpenSSL\_add\_ssl\_algorithms()SSL\_library\_init() #define SSLeay\_add\_ssl\_algorithms()SSL\_library\_init()

**12.3.2 初始化** **SSL 上下文环境变量函数** **SSL\_CTX\_new**

该函数用于初始化SSL CTX结构体，设置SSL 协议算法。可以设置SSL 协议的哪个版本， 以及客户端的算法或服务端的算法。该函数声明如下：

SSL\_CTX\*SSL\_CTX\_new(SSL METHOD\*meth);

其中，参数meth[in] 表示使用的是SSL 协议算法。OpenSSL 支持的算法如表12-1所示。

**表12-1** **OpenSSL 支持的算法**

|  |  |
| --- | --- |
| **函** **数** | **说** **明** |
| SSL\_METHOD\*SSLv2\_server\_method(); | 基于SSL V2.0协议的服务端算法 |
| SSL\_METHOD\*SSLv2 client\_method(); | 基于SSL V2.0协议客户端的算法 |
| SSL METHOD \*SSLv3\_server\_method(); | 基于SSL V3.0协议的服务端算法 |
| SSL METHOD \*SSLv3\_client\_method(); | 基于SSL V3.0协议的客户端算法 |
| SSL METHOD\*SSLv23\_server\_method(); | 同时支持SSL V2.0和3.0协议的服务端算法 |
| SSL\_METHOD \*SSLv23\_client\_method(); | 同时支持SSL V2.0和3.0协议的客户端算法 |
| SSL\_METHOD\*TLSvl\_server\_method(; | 基于TLS V1.0协议的服务端算法 |
| SSL\_METHOD\*TLSv1\_client\_method(); | 基于TLS V1.0协议的客户端算法 |



**Windows C/C++加密解密实战**

如果函数执行成功就返回SSL\_CTX结构体的指针，否则返回NULL。

**12.3.3** **释放SSL** **上下文环境变量函数SSL\_CTX\_free**

该函数用于释放SSL\_CTX结构体，该函数要和SSL\_CTX\_new配套使用，函数声明如下： void SSL\_CTX\_free(SSL\_CTX\*ctx)

其中，ctx[in]是已经初始化的SSL 上下文的SSL\_CTX结构体指针，表示SSL 上下文环境。

**12.3.4** **文件形式设置SSL** **证书函数SSL\_CTX\_use\_certificate\_file**

该函数以文件的形式设置SSL 证书。对于服务端，用来设置服务器证书；对于客户端， 用来设置客户端证书。函数声明如下：

int SSL CTX use certificate file(SSL CTX\*ctx,const char\*file,int type);

其中，参数 ctx[in]是已经初始化的SSL 上下文的SSL\_CTX结构体指针，表示 SSL 上下 文环境；file[in]表示证书路径；type[in]表示证书的类型，type 取值如下：

● SSL\_FILETYPE\_PEM:PEM格式的，即Base64 编码格式的文件。

● SSL\_FILETYPE\_ASN1:ASN1格式的，即DER 编码的文件。 如果函数执行成功就返回1,否则返回0。

**12.3.5** **结构体方式设置SSL** **证书函数SSL\_CTX\_use\_certificate**

该函数用于设置证书，函数声明如下：

int SSL\_CTX\_use\_certificate (SSL\_CTX\*ctx,X509\*x);

其中，参数 ctx[in]是已经初始化的SSL\_CTX结构体指针，表示SSL 上下文环境；X509[in] 表示数字证书。如果函数执行成功就返回1,否则返回0。

**12.3.6** **文件形式设置SSL** **私钥函数SSL\_CTX\_use\_PrivateKey\_file**

该函数以文件形式设置SSL 私钥，函数声明如下：

int SSL CTX use PrivateKey file(SSL CTX\*ctx,const char \*file,int type)

其中，参数ctx[in]是已经初始化的SSL 上下文的SSL\_CTX 结构体指针，表示SSL 上下 文环境；file[in]表示私钥文件路径；type[in]表示私钥的编码类型，支持的参数如下：

● SSL\_FILETYPE\_PEM:PEM格式的，即Base64 编码格式的文件。

● SSL\_FILETYPE\_ASN1:ASN1格式的，即DER 编码的文件。 如果函数执行成功就返回1,否则返回0。



第12章 SSL-TLS 编程

**12.3.7** **结构体方式设置SSL** **私钥函数SSL\_CTX\_use\_PrivateKey**

该函数以结构体方式设置SSL私钥，函数声明如下：

int SSL\_CTX\_use\_PrivateKey(SSL\_CTX\*ctx,EVP\_PKEY\*pkey);

其中，参数ctx[in]是已经初始化的SSL\_CTX 结构体指针，表示SSL 上下文环境；pkey[in] 是EVP\_PKEY 结构体的指针，表示私钥。如果函数执行成功就返回1,否则返回0。

**12.3.8** **检** **查SSL** **私钥和证书是否匹配函数SSL\_CTX\_check\_private\_key**

该函数检查私钥和证书是否匹配，该函数必须在设置了私钥和证书后才能调用，函数声明 如下：

int SSL\_CTX\_check\_private\_key(const SSL\_CTX\*ctx);

其中，参数ctx[in] 是已经初始化的SSL\_CTX 结构体指针，表示SSL 上下文环境。如果匹 配成功就返回1,否则返回0。

**12.3.9 创** **建** **SSL 结构函数SSL\_new**

该函数用于申请一个SSL 套接字，即创建一个新的 SSL 结构，用于保存TLS/SSL 连接的 数据。新结构继承了底层上下文ctx、连接方法(SSL v2/SSL v3/TLS vl)、选项、验证和超 时的设置。该函数声明如下：

SSL\*SSL\_new(SSL\_CTX\*ctx)

其中，参数 ctx[in] 表示上下文环境。如果函数执行成功就返回 SSL 结构体指针，否则返 回 NULL。

**12.3.10** **释** **放SSL** **套接字结构体函数SSL\_free**

该函数用于释放由SSL\_new 建立的SSL 结构体，在内部，该函数会减少SSL 的引用计数， 并删除SSL结构，如果引用计数已达到0,就释放分配的内存。该函数声明如下：

void SSL\_free(SSL\*ssl);

其中，参数 ssl[in] 表示要删除释放的 SSL 结构体指针。

**12.3.11 设置读写套接字函数SSL\_set\_fd**

该函数用于设置SSL套接字为读写套接字。该函数声明如下：

int SSL\_set\_fd(SSL\*s,int fd);

其中，参数s[in] 表示 SSL 套接字(结构体)的指针，fd 表示读写文件描述符。如果函数 执行成功就返回1,否则返回0。



Windows C/C++加密解密实战

**12.3.12 设置只读套接字函数SSL\_set\_rfd**

该函数用于设置SSL 套接字为只读套接字。该函数声明如下： int SSL\_set\_rfd(SSL\*s,int fd)

其中，参数s[in]表示SSL套接字(结构体)的指针，fd 表示只读文件描述符。如果函数 执行成功就返回1,否则返回0。

**12.3.13** **设置只写套接字函数** **SSL\_set\_wfd**

该函数用于设置SSL 套接字为只写套接字。该函数声明如下： int SSL set wfd(SSL\*s,int fd)

其中，参数s[in]表示 SSL 套接字(结构体)的指针，fd 表示只写文件描述符。如果函数 执行成功就返回1,否则返回0。

**12.3.14 启动TLS/SSL握手函数SSL\_connect**

该函数用于发起SSL 连接，即启动与TLS/SSL服务器的TLS/SSL握手，该函数声明如下： int SSL\_connect(SSL \*ssl);

其中，参数ssl[in]表示SSL套接字(结构体)的指针。如果函数执行成功就返回1,否则 返回0。

**12.3.15** **接** **受SSL** **连接函数SSL\_accept**

该函数用在服务端，表示接受客户端的 SSL 连接，类似于Socket编程中的accept函数。 该函数声明如下：

int SSL\_accept(SSL\*ssl);

其中，参数ssl[in]表示SSL 套接字(结构体)的指针。如果函数执行成功就返回1,表示 TLS/SSL握手已成功完成，已建立TLS/SSL连接。如果返回0,表示 TLS/SSL握手不成功， 但已被关闭，由TLS/SSL 协议的规范控制，此时可以调用函数SSL\_get\_error()找出原因。如 果返回值小于0,表示TLS/SSL握手失败，原因是在协议级别发生了致命错误，或者发生了 连接故障，此时可以调用函数SSL\_get\_error()找出原因。

**12.3.16** **获** **取** **对** **方** **的X509** **证** **书** **函** **数SSL\_get\_peer\_certificate**

该函数用于获取对方的X509证书。根据协议定义，TLS/SSL服务端将始终发送证书(如果 存在)。只有在服务端明确请求时，客户端才会发送证书。如果使用匿名密码，就不发送证书。

如果返回的证书不指示有关验证状态的信息，就使用SSL-get-verify-result检查验证状态。 该函数将导致X509对象的引用计数递增一，这样在释放包含对等证书的会话时，它不会被销 毁，必须使用X509\_free()显式释放X509对象。



第 1 2 章 SSL-TLS 编程

该函数声明如下：

X509\*SSL\_get\_peer\_certificate(const SSL\*ssl);

其中，参数sl[in] 表示 SSL 套接字(结构体)的指针。如果函数执行成功，就返回对方提 供的证书结构体的指针，如果返回NULL, 表示对方未提供证书或未建立连接。

12.3.17 向 TLS/SSL 连接写数据函数SSL\_write

该函数将缓冲区buf 中的 num 字节写入指定的SSL 连接，即发送数据。该函数声明如下： int SSL\_write(SSL\*ssl,const void \*buf,int num);

其中，参数 ssl[in] 表示SSL 套接字(结构体)的指针，buf 表示要写入的数据，num 表示 写入数据的字节长度。如果返回值大于0,表示实际写入的数据长度。如果返回值等于0,表 示写入操作未成功，原因可能是基础连接已关闭，此时可以调用 SSL\_get\_error()查明是否发生 错误或连接已完全关闭(SSL\_error\_ZERO\_return),SSLv2 (已弃用)不支持关闭警报协议， 因此只能检测是否关闭了基础连接。如果返回值小于0,表示写入操作未成功，原因要么是发 生错误，要么是调用进程必须执行某个操作，调用SSL\_get\_error()可以找出原因。

12.3.18 从TLS/SSL 连接中读取数据函数SSL\_Read

该函数尝试从指定的SSL 连接中读取num 字节到缓冲区buf。该函数声明如下： int SSL\_read(SSL\*ssl,void \*buf,int num);

其中，参数 ssl[in]表示SSL 套接字(结构体)的指针；buf[in] 指向一个缓冲区，该缓冲区 用于存放读到的数据；num 表示要读取数据的字节数。如果返回值大于0,表示读取操作成功， 此时返回值是从TLS/SSL 连接中实际读取到的字节数；如果返回值为0,表示读取操作未成 功，原因可能是由于对方发送“关闭通知”警报而导致完全关闭，在这种情况下，设置处于 SSL 关闭状态的SSL\_RECEIVED\_SHUTDOWN 标志也有可能，对方只是关闭了底层传输，而 关闭是不完整的，使用SSL\_get\_error() 函数可以获得错误信息，以查明是否发生错误或连接已 完全关闭(SSL\_ERROR\_ZERO\_RETURN)); 如果返回值小于0,表示读取操作未成功，原 因可能是发生错误或进程必须执行某个操作，此时可以调用SSL\_get\_error() 找出原因。

1 2 . 4 准 备SSL通信所需的证书

由 于SSL 网络编程需要用到证书，因此我们需要搭建环境建立CA, 并签发证书。

12.4.1 准备实验环境

严格来讲，应该准备三台安装Windows 系统的计算机，CA 端一台、服务端一台、客户端



**Windows C/C++加密解密实战**

一台，然后在服务端生成证书请求文件，再复制到CA 端去签发，再把签发出来的服务端证书 复制到服务端保存好。同样，客户端也是先生成证书请求文件，但考虑到某些同学的机器性能 或者没有那么多台计算机，所以我们就用一台物理机来完成所有证书签发工作。对于实验而言 方便一些，避免要在多个Windows 下安装VC2107 和 OpenSSL。

**12.4.2** **熟悉CA** **环境**

我们的CA 准备通过OpenSSL来实现，而编译安装OpenSSL1.0.2m 后，基本的CA 基础 环境也就有了。在C:lmyOpenssoutssl 下有一个配置文件 openssl.cnf。我们可以直接通过该配 置文件来熟悉这个默认的CA 环境。

要手动创建CA 证书，就必须首先了解OpenSSL中关于CA 的配置，配置文件的位置在 C:lmyOpensslouts\openssl.cnf。我们通过Windows下的编辑软件(比如Notepad++或UltraEdit 等)可以查看其内容。

**12.4.3** **创建所需要的文件**

根据CA 配置文件，一些目录和文件需要预先建立好。首先在C:\myOpensslout\bin\下新建 一个文件夹demoCA, 再 在demoCA下建立子文件夹 newcerts, 接着在demoCA下新建两个文 本文件 index.txt 和 serial, 并 用Notepad++打开serial 后输入01,然后保存并关闭。如果不提 前创建这两个文件，那么在生成证书的过程中会出现错误。

因为openssl.exe位于C:\myOpensslout\bin\下，所以我们要在C:\myOpensslout\bin\下新建 文件夹demoCA。

**12.4.4 创建根CA 的证书**

首先在物理主机上构造根CA 的证书。因为没有任何机构能够给根CA 颁发证书，所以只 能根CA 自己给自己颁发证书。首先要生成私钥文件，私钥文件是非常重要的文件，除了自己 以外，其他任何人都不能获取。所以在生成私钥文件的同时最好修改该文件的权限，并且采用 加密的形式生成。

我们可以通过执行OpenSSL中 的genrsa 命令生成私钥文件，并采用DES3 的方式对私钥 文件进行加密，过程如下：

(1)生成CA 根证书私钥

在命令行下进入 C:\myOpensslout\bin\后执行 OpenSSL程序，然后在OpenSSL提示符下输 入命令：

genrsa -des3 -out root.key 1024

其中，genrsa表示采用RSA 算法生成根证书私钥；-des3表示使用3DES给根证书私钥加密； 1024表示根证书私钥的长度，建议使用2048,越长越安全。命令genrsa用来生成1024位的RSA 私钥，并在当前目录下自动新建一个root.key,私钥就保存到该文件中。在命令中，私钥用3DES



**第** **1** **2** **章** **SSL-TLS** **编程**

对称算法来保护，所以我们需要输入保护口令，这里输入123456,如图12-5所示。



区C:NTindos\systo32\adx-p1. 回 ×

c:vyOpensSLout ybin>opengp1.exe

opans8L>gonrsa -des3 -out root.key 1024

Generating RSA private key,1024 hit long nodulus

...

565537 (0xd881>

Enter pa=s phrase for root.key:

einsSig -Enter paOs phrase for root.key:



c:WsersAdninistrator>cdC:nyOpensSLouthin



图12-5

此时，如果到C:\myOpensslout\bin\ 下查看，可以发现多了一个root.key 文件，这就是我们 加过密的私钥文件，其格式是Base64 编码的PEM 格式文件。

(2)生成根证书请求文件

下面可以准备生成根证书了，有两种方式，如果我们的根证书需要别的签名机构来签名， 就需要先生成根证书签名请求文件，格式为.csr, 然后拿这个签名请求文件给该签名机构，让 其帮我们签名，签名完后，会返回一个.crt 格式的证书。生成证书请求文件的命令如下：

req -new -key root.key -out root.csr

其中，req 命令用来生成证书请求文件，注意生成证书请求文件需要用到私钥；-key 这里 需要指向上一步生成的根证书私钥；-out 这里会生成我们的根证书签名请求文件。

如果不想这样麻烦，可以自签根证书。这里我们就采用自签根证书的方法。

(3)生成CA 的自签证书

要生成自签证书，直接利用私钥即可。在OpenSSL 提示符下输入如下命令： req -new-x509-key root.key -out root.crt

该命令执行后，首先会要求输入root.key的保护口令(这里是123456),然后会要求输 入证书的信息，比如国家名、组织名等，如图12-6所示。



TcAvmratad … a … 回

ceviyOpegsSlout shin>opens1.ex 日

bpenssLOr⁴aov-5 的 -key Faot.kay -out root.ert Entar pa phease for root.key:

o are aoit to aaked to enter infornation Chat wi11 ie in into youp eertificaterequest.

hat you are ahout to enter chat is cal1eda Distingutiahed There ar uite 。feu field hut you can 1Ave sone hank

oenlds tthere v1 he kdefanltlalu

ountry Mone (2 lettor eode>[AU1:CH

state ar Provinee Msre 1 nan)[sone-State1:JIAHGsu oekiity e(eg.city)tikNJIH

Organzat ion Hne (g,conpang Interuet viagits Pty Ltdi:coH

oin1 ns.g.[vvsEP胸QRP0rcon YOUR nane) :HYSENUER

oranssi> 

oryantationahit Mon(esroeetion)-7:N

×

图12-6



**Windows C/C++加密解密实战**

此时，如果到C:\myOpenssloutlbin\下查看，可以发现多了一个root.crt 文件，这就是我们 的根证书文件。有了根证书，我们就可以为服务端和客户端签发出它们的证书了。同样，首先 要在两端分别生成证书请求文件，然后到CA 去签发出证书。

**12.4.5** **生成服务端的证书请求文件**

生成证书请求需要用到私钥，所以先要生成服务端的私钥。在OpenSSL 提示符下输入如 下命令：

genrsa-des3 -out server.key 1024

我们用了3DES 算法来加密保存私钥文件server.key,该命令执行过程中会提示输入3DES 算法的密码，这里输入123456。执行后，会在C:\myOpenssoutbin 下看到server.key, 这个文 件就是服务端的私钥文件。

然后，可以准备生成证书请求文件，在OpenSSL 提示符下输入如下命令： req -new-key server.key -out server.csr

在命令执行过程中，首先要求输入3DES的密码来对server.key 解密，然后生成证书请求 文件 server.csr, 生成证书请求文件同样需要输入一些信息，比如国家、组织名等。注意输入 的组织信息要和根证书一致，这里都是COM, 如图12-7所示。



**C:Windors\syste32\ed.exe-opensl.ex**

**OpensSL>rer -nav-key server,key -out** seroer.csr Enter pass pherase for serUer,key:

You are alout to be ackod to enter infornation that vill he inc into your certificate request.

Whatyo are ahout to entor ia what is called a Diat inguiohedk

r

Country Hane (21etter code)[AUJ:CH

Stato or Province Mane (ful1 nane)(Song-State 1:JIANGSU

Locality Hane (eg,city)[]:NAKJIN

Oranization Mann (ey,conpany)[Internet Vidgita Pty Ltd1:CoH

Organizational Unit Hone (eg,uect ion)[]:MYUNIT

Connon Hane (e.g.server FQDN ar YOUR nane)[1:SRU Enail Address :SHUEQ9.COH

Please enter the folloving 'extra'attributes to he cent vith your certif icate request

A challenge paasword O:123456



in optional conpany nane [1: ppanssL>

回 区 日

图12-7

此时，如果到C:lmyOpenslout bin\下查看，可以发现多了一个 server.csr 文件，这就是我 们的服务端的证书请求文件，有了它，就可以到CA 那里签发证书了。

**12.4.6** **签发出服务端证书**

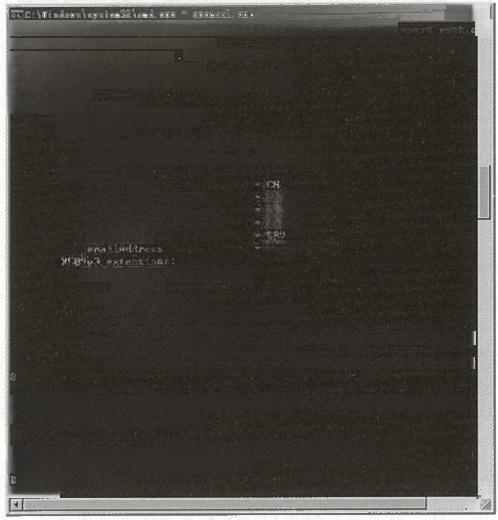
在OpenSSL提示符下输入如下命令：

ca -in server.csr -out server.crt -keyfile root.key -cert root.crt -days 365 -config../ssl/openssl.cnf



**第** **1** **2** **章** **SSL-TLS 编** **程**

其 中 ，ca 命令用来签发证书；-in表示输入CA 的文件，这里需要输入的是证书请求文件 server.csr;-out 表示CA 输出的证书文件，这里输出的是server.crt;-days 表示所签发的证书 有效期，这里是365天。该命令执行过程中，会首先要求输入 root.key的保护口令，然后要求 确认两次信息，输入y 即可，如图12-8所示。



ox

OpensSSL>ea-in gerver.cir -out erver.crt -lkayrile root.key

365 -confi.7l/opens⁰1.cnf

Using configuration fron - lapenssl.cnf Enter pass phrace for ront.key:

chack tht the rqueat mtahei tha gignaturo

signature ok

cartificaoerileHtaihler:2(a

0nIidity

Hot lefore:Apr 100:35:072020 cn

Hot Aftr年Apr 1800:35:072021 cHI suhjeat:

countrgHane

ctateOrIrov incaHane

organlzat ionHane

HVUNIT

coinonHane

siUeQ9.coH

X50903 nnis Gomstraints:

cnimLSE

Hetecape Comnent:

0penssL Generated Certif icate X50903 suhsect Key Identifier:

X50BtFh:9:3K7e:⁶:I4d0e:DtI:8i:69:94:P9:00=3E⁸40833:20

keyid:E1: 日9890880:09:B:4E:13:92:4C:A1:50:82EE⁸F62:1

Cartif icate ig to he oertif ied unti1 Apr 188:35:072021 GhT (365 days) Siun the cartif ioato?[g/nliy

out of 1 certif icate requests cart Ified.conmit?[y/nly Urito out databaae vith 1 neu entrias

ta Dase Updated

ohensSL>

2

ohmlatlonhTnR

JIANOSU COH



图12-8

此时，如果到C:\myOpensslout\bin\ 下查看，可以发现多了一个 server.crt 文件，这就是我 们的服务端的证书文件。

**12.4.7** **生** **成** **客** **户** **端** **的** **证** **书** **请** **求** **文** **件**

生成证书请求需要用到私钥，所以先要生成服务端的私钥。在OpenSSL 提示符下输入如 下命令：

genrsa -des3 -out client.key 1024

我们用了3DES 算法来加密保存私钥文件client.key,该命令执行过程中，会提示输入3DES 算法的密码，这里输入123456。执行后，会在C:\myOpensslout\bin下看到client.key,这个文 件就是服务端的私钥文件。

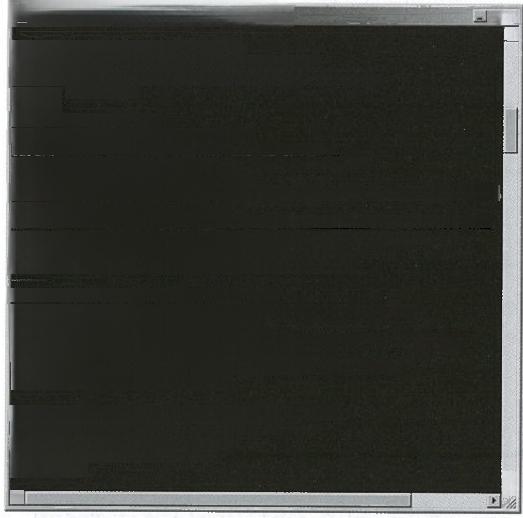
然后，可以准备生成证书请求文件，在OpenSSL 提示符下输入如下命令： req-new-key client.key -out client.csr

在命令执行过程中，首先要求输入3DES 的密码来对client.key解密，然后生成证书请求 文件client.csr, 生成证书请求文件同样需要输入一些信息，比如国家、组织名等。注意输入的



**Windows C/C++加密解密实战**

组织信息要和根证书一致，这里都是COM, 如图12-9所示。



**TCVadoa\sts2\da a**

C:vnyOpensSLout \bin>opongn1.0xe

OpenSSL>genraa -des3 -out client.key 1024

Generating RSAprivate key,1024 bit long modulus

..+

is 65537(Gx18801)

Enter passphrase for client.key:

Werifying -Enter pass phrase for client.key:

OpenSSL>rey -new -key client.key -out client.csr Enter pass phrase for client.key:

What you are about to gnter io what is called a Distinguiohed Hane There are quite a few ficldshut you can leave cone blank

For sone fields there wil1 he a dofault value,

If you enter.,the field uil1 he left blank.

Country Hane(2 letter code)[AU1:CN

li Pice (ay)(I)one-State l:JIAHGSU

Organization Hane (es,conpany)[Internet Widgita Pty Ltd1:COH

Organizational Unit Hane (eg.cection)[1:HYUNIT

Connon Hane (0.g.server FQDH or YOUR nane)[1:CLIENT Enail nddress r1:CLI ENTBQ⁹.COM

Pleasn enter the falloving 'oxtra'attributen

to be sent with your certificate requent

A challenge passord [1:123456 0n optionalconpany name []:

OpenSSL>

A

You are about to he asked to enter infornationthat vill be incorpo into your certificato request.

回 × A



图12-9

此时，如果到C:\myOpensslout\bin\ 下查看，可以发现多了一个client.csr 文件，这就是我 们的服务端的证书请求文件，有了它，就可以到CA 那里签发证书了。

**12.4.8** **签发客户端证书**

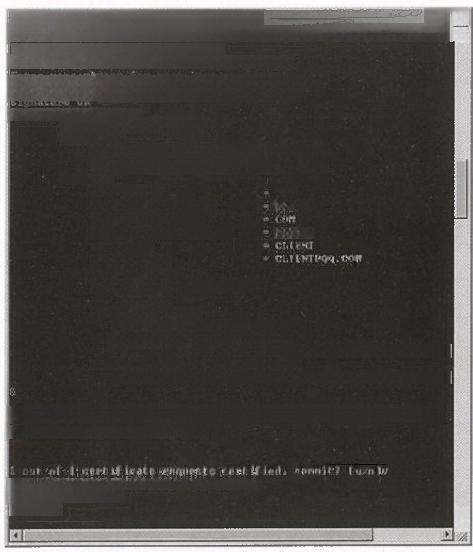
在OpenSSL 提示符下输入如下命令：

ca -in client.csr -outclient.crt -keyfile root.key -cert root.crt -days 365-config ../ssl/openssl.cnf

其 中 ，ca 命令用来签发证书；-in表示输入 CA 的文件，这里需要输入的是证书请求文件 server.csr;-out 表示CA 输出的证书文件，这里输出的是client.crt;-days 表示所签发的证书有 效期，这里是365天。该命令执行过程中，会首先要求输入root.key 的保护口令，然后要求确 认两次信息，输入y 即可，如图12-10所示。



**第** **1** **2** **章** **SSL-TLS 编程**



**区C:NTindosyt2\ead.-pnscl.o** 日 回 冈

OponsSL>an In eliont.car -ont client.crt -hoyfile root.key -enrt 口

Ug co-figut io.s0..opangs1.cnf

k tht th t matele the aimatre

Cartif icato Details:

Serial HnhrE.30

d

Hot lefore fpr 18 ::392828 cHI

Hot After :Apr 1808:08:392021 GHT

oIoct

GountrvH

atataOrProv inccHane

arganizationHane

organizntionalunitHane

conho

anaiIAddre

X58903 oxtensiong:

858903 Baric Constrainta;

CA:FALSE

Hetscape Corment:

OpenS8L,Generated Certif

X50901 samhject Key Identif ieri

858D9::3f2u:4t:i::2N5:i:e::8T=E⁹:97:7D:82:

kayid=E1:BE:89:98=88:A⁹=8A:4E:13:92:4CEA1:58:82:E8:

eart if icato is to he aortif iod until Apr1888:8:392021 cm(365 Sion the cortIficato?[y/n1:y

Write out datohase with 1 new ent ro

Data Bauo Updated

opensSL>



CN

JIAHGSIU

HYUNII

icate

图12-10

此时，如果到C:\myOpensslout\bin\ 下查看，可以发现多了一个client.crt 文件，这就是我 们的客户端的证书文件。

至此，服务端和客户端证书全部签发成功，双方有了证书就可以进行SSL 通信了。

**12.5 实战SSL** **网络编程**

我们的程序是一个安全的网络程序，分为两部分：客户端和服务端。我们的目的是利用 SSL/TLS 的特性保证通信双方能够互相验证对方的身份(真实性),并保证数据的完整性、 私密性，这三个特性是任何安全系统中常见的要求。

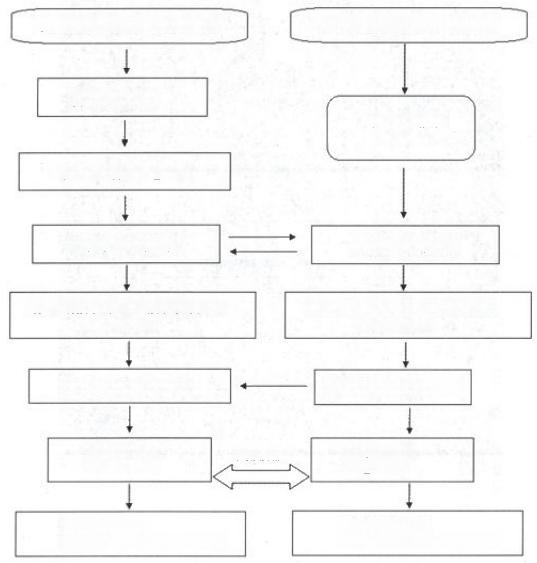
对程序来说，OpenSSL 将整个SSL 握手过程用一对函数体现，即客户端的SSL\_connect 和服务端的SSL\_accept, 而后的应用层数据交换则用SSL\_read 和 SSL\_write 来完成。

SSL 通信的一般流程如图12-11所示。



Windows C/C++加密解密实战

服务端 客户端



OpenSSL初始SSL\_library\_init() OpenSSL初始SSL\_library\_init()

设置证书和私钥

创建SSL套接字 SSL\_nem()

创建SSL套接字SSL\_new()

正常的TCP连接过程 正常的TCP连接过程

将TCP套接字与SSL套接字联系起来 SSL\_set\_fd()

接受连接SSL\_accetp() 发起连接SSL\_connectO

|  |  |  |
| --- | --- | --- |
| SSL\_read() SSL\_write() | 交换数据 | SSL\_write() SSL\_read() |
| SSL\_shutdown()  SSL\_free();SSL\_CTX\_free() |  | SSL\_shutdown()  SSL\_free();SSL\_CTX\_free( |

将TCP套接字与SSL套接字联系起来 SSL\_set\_fd()

图12-11 基本上，编程流程就是按照这个模型来进行的。

【例12.1】SSL 服务端和客户端通信

(1)首先创建服务端工程。打开VC 2017,新建一个控制面板工程，工程名是sslserver。 ( 2 ) 在VC 中打开sslserver.cpp, 并输入代码如下：

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*SSL/TLS服务端程序WIN32版(以demos/server.cpp为基础)

\*需要用到动态连接库libeay32.dll、ssleay.dll

\*同时在setting中加入ws2\_32.lib、libeay32.lib、ssleay32.lib

\*以上库文件在编译 OpenSSL 后可在out32dll目录下找到 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "pch.h"

#include<stdio.h>

#include<stdlib.h>

#include <memory.h> #include <errno.h>

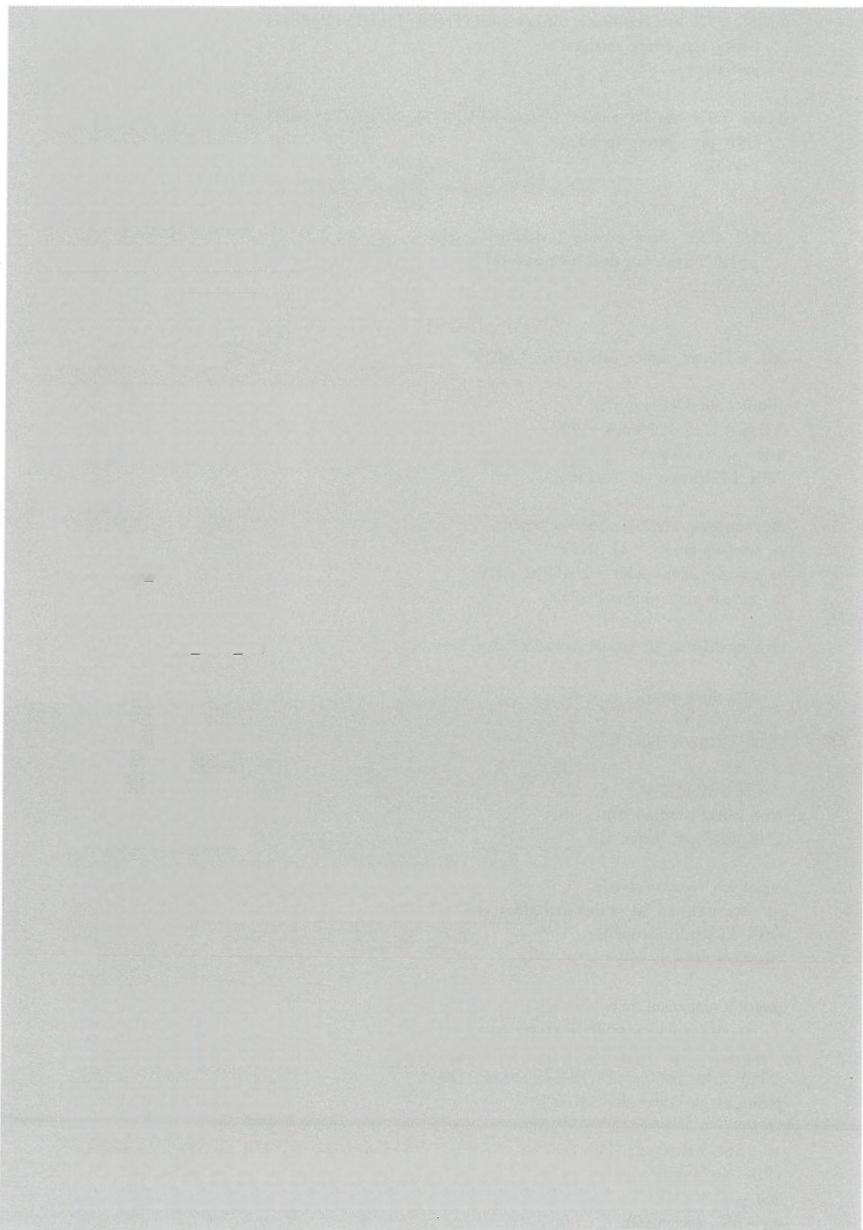
#include <sys/types.h>

#include <winsock2.h>

#include "openssl/rsa.h"



**第12章** **SSL-TLS** **编程**

#include "openssl/crypto.h" #include "openssl/x509.h" #include "openssl/pem.h" #include "openssl/ssl.h"

#include "openssl/err.h"

#pragma comment(lib,"libeay32.lib") #pragma comment(lib,"ssleay32.lib") #pragma comment(lib,"ws2\_32.1ib")

/\*所有需要的参数信息都在此处以#define的形式提供\*/

#define CERTF "server.crt" #define KEYF "server.key" #define CACERT "root.crt"

#define PORT 1111

/\*服务端的证书(需经CA 签名)\*/ /\*服务端的私钥(建议加密存储)\*/ /\*CA的证书\*

/\*准备绑定的端口\*/

#define CHK\_NULL(x)if((x)==NULL)exit(1)

#define CHK\_ERR(err,s)if((err)=-1){perror(s);exit(1);}

#define CHK\_SSL(err)if((er)==-1){ERR\_print\_errors\_fp(stderr);exit(2);}

int main() {

int err;

int listen\_sd; int sd;

struct sockaddr\_in sa\_serv; struct sockaddr in sa cli; int client\_len;

SSL CTX\*ctx SSL\* ssl;

X509\* client\_cert;

char\* str;

char buf[4096];

const SSL\_METHOD\*meth WSADATA wsaData;

if(WSAStartup(MAKEWORD(2,2),&wsaData)!=0){ printf("WSAStartupO)fail:%d\n",GetLastError(); retum -1;

}

SSL\_load\_error\_strings();

OpenSSL\_add\_ssl\_algorithms(); meth=TLSvl\_server\_method();

/\*为打印调试信息作准备\*/ /\*初始化\*/

/\*采用什么协议(SSL v2/SSL v3/TLSvl)在此指定\*

ctx=SSL\_CTX\_new(meth); CHK\_NULL(ctx);

SSL\_CTX\_set\_verify(ctx,SSL\_VERIFY\_PEER,NULL); /\*验证与否\*/

SSL\_CTX\_load\_verify\_locations(ctx,CACERT,NULL);/\*若验证，则放置CA 证书\*/



**Windows** **C/C++加密解密实战**

if (SSL\_CTX\_use\_certificate\_file(ctx,CERTF,SSL\_FILETYPE\_PEM)<=0){

ERR\_print\_errors\_fp(stderr) exit(3);



if(SSL\_CTX\_use\_PrivateKey\_file(ctx,KEYF,SSL\_FILETYPE\_PEM)<=0){

ERR\_print\_errors\_fp(stder); exit(4);

}

if(ISSL\_CTX\_check\_private\_key(ctx)){

printf("Private key does notmatch the certificate public key\n"); exit(5);

}

SSL\_CTX\_set\_cipher\_list(ctx,"RC4-MD5");

printf("I am ssl-server\n");

/\*开始正常的TCP Socket 过 程 . . . … . . .

listen\_sd=socket(AF\_INET,SOCK\_STREAM,0); CHK\_ERR(listen\_sd,"socket");

memset(&sa\_serv,'0',sizeof(sa\_serv); sa\_serv.sin\_family=AF\_INET;

sa\_serv.sin\_addr.s addr=INADDR\_ANY sa\_serv.sin\_port=htons(PORT);

err=bind(listen\_sd,(struct sockaddr\*)&sa\_serv

sizeof(sa\_serv);

CHK\_ERR(err,"bind");

/\*接受TCP 链接\*/

err=listen(listen\_sd,5); CHK\_ERR(err,"listen")

client\_len=sizeof(sa\_cli);

sd=accept(listen\_sd,(struct sockaddr\*)&sa\_cli,&client\_len);

CHK\_ERR(sd,"acept") closesocket(listen\_sd);

printf("Connection from %lx,port %x\n" sa\_cli.sin\_addr.s\_addr,sa\_cli.sin\_port)

/\*TCP连接已建立，进行服务端的SSL 过程\*/ printf("Begin server side SSL\n")

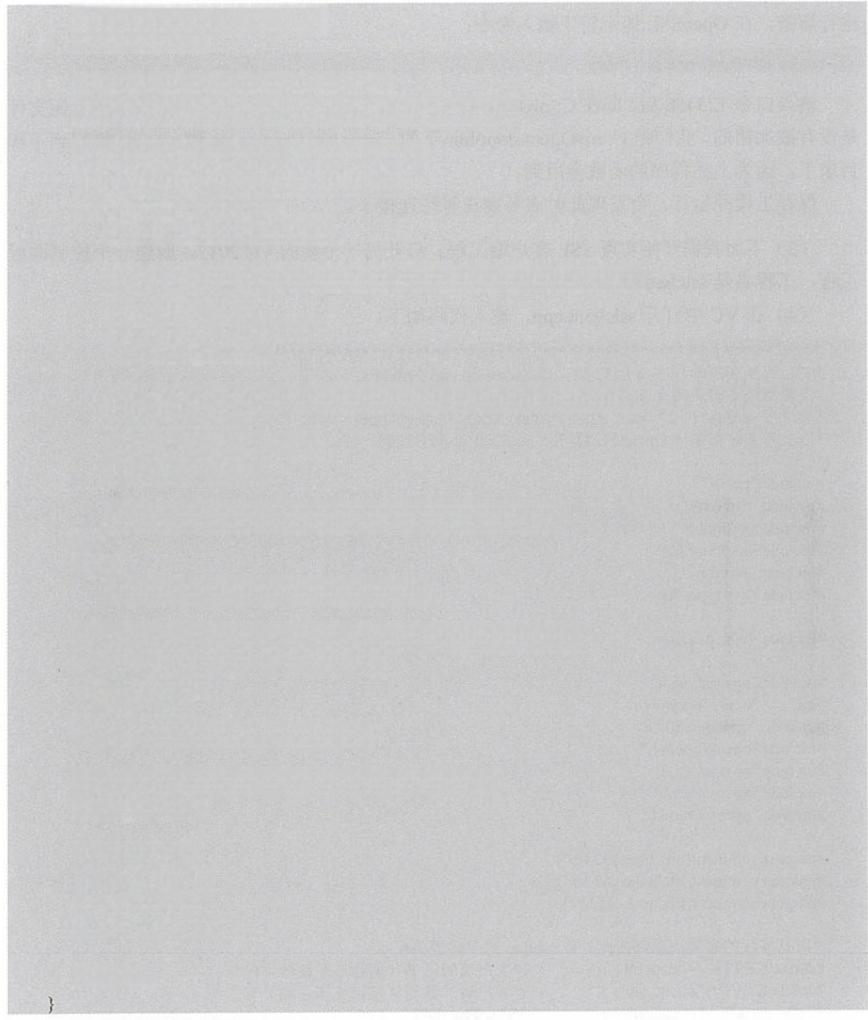
ssl=SSL new(ctx) CHK\_NULL(ssl);

SSL set fd(ssl,sd)

err=SSL\_accept(ssl);



第12章 SSL-TLS 编程

printf("SSL\_accept finished\n") CHK\_SSL(err);

/\*打印所有加密算法的信息(可选)\*/

print("SSL connection using%shn",SSL\_get\_cipher(sl);

/\*得到服务端的证书并打印信息(可选)\*/ client\_cert =SSL\_get\_peer\_certificate(ssl);

f(client\_cert !=NULL){

printf("Client certificate:\n");

str=X509\_NAME\_oneline(X509\_get\_subject\_name(client\_cert),0,0); CHK\_NULL(str);

printf("\t subject:%s\n",str); OPENSSL\_free(str);

str=X509\_NAME\_oneline(X509\_get\_issuer\_name(client\_cert),0,0); CHK\_NULL(str)

print("t issuer:%s\n",str); OPENSSL\_free(str)

X509\_free(client\_cert);/\*若不再需要，则需将证书释放\*/ }

else

printf("Client does not have certificate.\n");

/\*数据交换开始，用SSL\_write 、SSL\_read 代替write 、read\*/ err=SSL\_read(ssl,buf,sizeof(buf)-1);

CHK\_SSL(err); buf[err]="0';

printf("Got %d chars:'%s'n",err,buf)

err=SSL\_write(ssl,"I hear you.",strlen("I hear you."); CHKSSL(err)

/\*收尾工作\*/ shutdown(sd,2); SSL\_free(ssl);

SSL\_CTX\_free(ctx); returm 0;

打开“test属性页”对话框，在“配置属性”→“CIC++”→“常规”→ “附加包含目录” 右边添加 C:\openssl-1.0.2mlinc32,然后保存并关闭工程属性对话框。接着，把 C:myOpenssloutlibl 下的libeay32.ib 和 ssleay32.lib 放到工程目录下，把C:lmyOpensslout bin 下的 libeay32.dll 和 ssleay32.dll 放到解决方案的 Debug 目录下，即与生成的 .exe 文 件 在 同 一 目 录下。

由 于 程 序 中 使 用 的server.key必须处于已经解密的状态，因此我们要对加密过的server.key



**Windows C/C++加密解密实战**

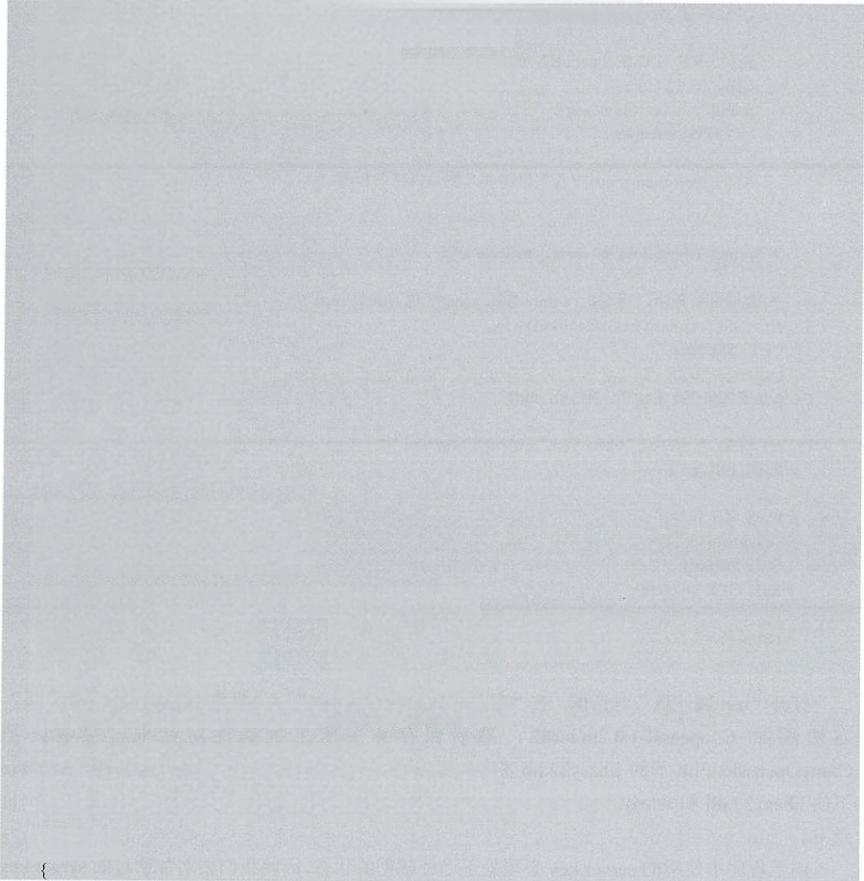
进行解密，在OpenSSL提示符下输入命令： rsa -in server.key -out server.key

输入口令123456后，即在C:myOpensslout\bin生成新的server.key,此时的这个私钥文件 是没有被加密的。我们把 C:\myOpensslout\bin\下的server.key 、server.crt和root.crt 复制到工程 目录下，因为上述程序的函数会用到。

保存工程并运行，会发现此时服务端在等待连接了。

(3)下面我们开始实现SSL 客户端工程。打开另一个新的VC 2017,新建一个控制面板 工程，工程名是sslclient。

( 4 ) 在VC 中打开sslclient.cpp, 输入代码如下：

/\***\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*** \*SSL/TLS 客户端程序WIN32 版 ( 以demos/cli.cpp为基础)

\*需要用到动态连接库libeay32.dll 、ssleay.dl

\*同时在setting 中加入ws2\_32.lib 、libeay32.lib 、ssleay32.lib

\*以上库文件在编译OpenSSL 后可在 out32dll目录下找到 \*/

#include "pch.h"

#include <stdio.h>

#include <stdlib.h>

#include <memory.h> #include<errno.h>

#include<sys/types.h>

#include<winsock2.h>

#include "openss/rsa.h"

#include "openss/crypto.h" #include "openss/x509.h" #include "openssl/pem.h"

#include "openssVssL.h"

#include "openssl/err.h"

#include "openssl/rand.h"

#pragma comment(lib,"libeay32.lib") #pragma comment(lib,"ssleay32.lib") #pragma comment(lib,"ws2\_32.lib")

/\*所有需要的参数信息都在此处以#define 的形式提供\*/

#define CERTF "client.crt" #define KEYF "client.key" #define CACERT "root.crt" #define PORT 1111

#define SERVER\_ADDR" <127.0.0.1>"

/\*客户端的证书(需经CA 签名)\*/ /\*客户端的私钥(建议加密存储)\*/ /\*CA的证书\*/

/\*服务端的端口\*/

/\*服务段的 IP 地址\*/

#define CHK\_NULL(x)if((x)—=NULL)exit(-1)

#define CHK\_ERR(err,s)if(err)=-1){perror(s);exit(-2);}

#define CHK\_SSL(er)if(err)-=-1){ERR\_print\_errors\_fp(stderr);exit(-3);}

int main()

第12章 SSL-TLS 编程

int err; int sd: struct sockaddr\_in sa; SSL\_CTX\* ctx; SSL\* ssl;

X509\* server\_cert;

char\* str;

char buf[4096];

const SSL\_METHOD \*meth;

int seed\_int[100]; 1\*存放随机序列\*

WSADATA wsaData

if (WSAStartup(MAKEWORD(2,2),&wsaData)!=0) {

printf("WSAStartup()fail:%d\n",GetLastError() retum -1;



\*初始化\*/

OpenSSL add ssl algorithms(0); /\*为打印调试信息作准备\*/

SSL\_load\_error\_strings();

1\*采用什么协议(SSL v2/SSL v3/TLS v1)在此指定\* meth=TLSv1\_client \_method0;

/\*申请SSL 会话环境\*/

ctx=SSL\_CTX\_new(meth); CHK\_NULL(ctx);

/\*验证与否，是否要验证对方\*/

SSL\_CTX\_set\_verify(ctx,SSL\_VERIFY\_PEER,NULL); /\*若验证对方，则放置CA 证书\*

SSL\_CTX\_load\_verify\_locations(ctx,CACERT,NULL);

/\*加载自己的证书\*

if(SSL\_CTX\_use\_certificate file(ctx,CERTF,SSL\_FILETYPE\_PEM)<=0)

{

}

ERR\_print\_errors exit(-2);

fp(stderr);

/\*加载自己的私钥，以用于签名\*

if(SSL\_CTX\_use\_PrivateKey\_file(ctx,KEYF,SSL\_FILETYPE\_PEM)<=0) {

ERR\_print\_errors\_fp(stderr); exit(-3);



\*调用了以上两个函数后，检验一下自己的证书与私钥是否配对\*

if(!SSL\_CTX\_check\_private\_key(ctx) {

printf("Private key does not match the certificate public keyln"); exit(-4);

**Windows** **C/C++加密解密实战**

/\*构建随机数生成机制，WIN32 平台必需\*/

srand(unsigned)time(NULL); for (int i=0;i<100;i++)

seed\_int[i]=rand();

RAND\_seed(seed\_int,Sizeof(seed\_int);

printf("I am ssl-client\n");

/\*开始正常的TCP Socket过程.....……......-/

sd=socket(AF\_INET,SOCK\_STREAM,0); CHK\_ERR(sd,"socket")

memset(&sa,"\0',sizeof(sa); sa.sin\_family=AF\_INET;

sa.sin\_addr.s\_addr=inet\_addr(SERVER\_ADDR);/\* 服务器 IP 地址\*/

sa.sin\_port =htons(PORT) /\*服务器端口\*/

err =connect(sd,(struct sockaddr\*)&sa,sizeof(sa)) CHK\_ERR(err,"connect");

/\*TCP 链接已建立，开始SSL 握手过程 . . . … . … · /

printf("Begin SSL negotiation \n");

/\*申请一个SSL 套接字\* ssl=SSL\_new(ctx)

CHK\_NULL(ssl);

/\*绑定读写套接字\*/ SSL\_set\_fd(ssl,sd)

err=SSL\_connect(ssl) CHK SSL(err)

/\*打印所有加密算法的信息(可选)\*/

printf("SSL connection using %s\n",SSL\_get\_cipher(sl);

/\*得到服务端的证书并打印信息(可选)\*/ server\_cert=SSL\_get\_peer\_certificate(ssl);

CHK\_NULL(server\_cert);

printf("Server certificate:ln")

str=X509\_NAME\_oneline(X509\_get\_subject\_name(server\_cert),0,0); CHK\_NULL(str);

printf("t subject:%sìn",str) OPENSSL free(str)

str=X509\_NAME\_oneline(X509\_get\_issuer\_name(server\_cert),0,0); CHK\_NULL(str);

printf("\t issuer:%sìn",str); OPENSSL\_free(str);

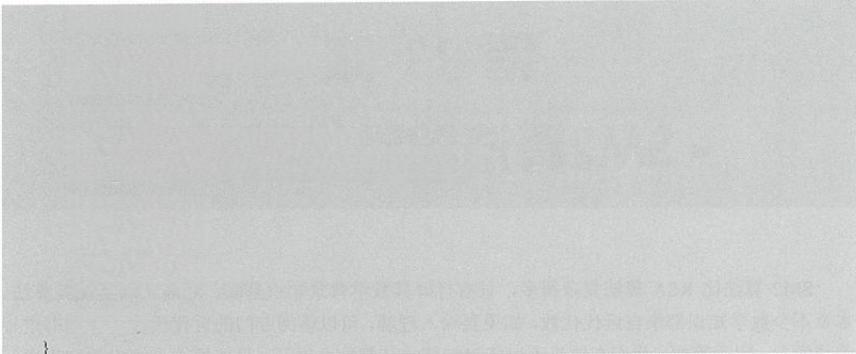
X509\_free(server\_cert);/\* 若不再需要，则需将证书释放\*/

/\*数据交换开始，用SSL\_write 、SSL\_read 代替write 、read\*/ printf("Begin SSL data exchange\n")

err =SSL write(ssl,"HelloWorld!",strlen("HelloWorld!"))



**第12章** **SSL-TLS** **编程**



CHK\_SSL(err)

err=SSL\_read(ssl,buf,sizeof(buf)-1); CHK\_SSL(err);

buf[err]=10'

printf("Got%d chars:'%s'n",err,buf)

SSL\_shutdown(ssl);/\* 发送 SSL/TLS 关闭通知\*/

/\*收尾工作\*/

shutdown(sd,2)

SSL\_free(ssl);

SSL\_CTX\_free(ctx);

return 0;

打开“test属性页”对话框，在“配置属性”→“CIC++”→“常规”→“附加包含目录” 右 边 添 加 C:lopenssl-1.0.2minc32, 然后保存并关闭工程属性对话框。接着，把 C:myOpenssloutlib\下的libeay32.ib 和 ssleay32.lib 放到工程目录下，把 C:lmyOpenssloutlbin 下的libeay32.d11和ssleay32.d 放到解决方案的Debug目录下，即与生成的.exe 文件在同一目 录下。

由于程序中使用的client.key必须处于已经解密的状态，因此我们要对加密过的client.key 进行解密，在OpenSSL提示符下输入命令：

rsa -in client.key-out client.key

输入口令123456后，即在C:myOpenssloutlbin生成新的client.key,此时的这个私钥文件 是没有被加密的。我们把 C:myOpensslout bin\下的client.key、client.crt和root.crt复制到工程 目录下，因为上述程序的函数会用到。

保存工程并运行，会发现此时和服务端能通信了，并且服务端打印出了服务端的证书，如

山



**Oierosoft** **Viso** **Stadio调试控制台** 回 ×

I an ssl-client

Begin S8L negotiation

SsL connection asing RC4-HD5 Server certificate:

 1gauer:C-CHSI-JIANGSU/L-HANJIH/O-Con/0

d hSRUBe00CO

Begin S8L data exchange

Got 11 chars:'I hear you.

D

subject:C-CH/8T-JIAMGSU/0-COM/OU-HYUHIT



图12-13



图12-12和图12-13所示。

**国** **icrosoft** **Visual** **Stadi。调试控钢台**

I am ssl-server

Connection fron 180007f,port fei1

Begin server side SSL SSL\_accept finished

ssL connection using RC4-MD5 Client certif icate:

subject:/C=CH/SI=JIANGSU/0-COH/0 eQ⁹.coH issuer:C=CH/ST=JIAHGSU/L=NANJIN ess=MYSERUERCQQ.COM

Got 12 chars:'Hello World!'



图12-12

区