第32卷 第16期 计 算 机 工 程 2006年8月
Vol.32 № 16 Computer Engineering August 2006
·开发研究与设计技术· 文章编号：1000—3428(2006)16—0283—03 文献标识码：A 中图分类号：TP309

一种嵌入式SSL协议栈的设计

罗建超[1]，俸 皓[2]，罗 蕾[1]
(1. 电子科技大学计算机学院，成都 610054；2. 桂林电子工业学院计算机系，桂林 541004)

摘 要：随着嵌入式技术的广泛应用，保障嵌入式应用的通信安全显得越来越必要。然而在嵌入式领域，以产品形式独立存在的SSL却是空白。该文给出了一种能满足嵌入式安全应用要求的SSL协议栈设计方案，提出了一种具有通用性的嵌入式网络体系结构模型。
关键词：嵌入式应用；SSL协议栈；通信安全

A Design of Embedded SSL Protocol Stack

LUO Jianchao[1], FENG Hao[2], LUO Lei[1]
(1. School of Computer Sci. & Eng., Univ. of Electronic Sci. & Tech. of China, Chengdu 610054;
2. Dept. of Computer Science, Guilin Univ. of Electronic Tech., Guilin 541004)

【Abstract】 As the embedded technology has been widely applied, to ensure the communication security of embedded applications becomes more and more necessary. However, independent SSL product for embedded applications has not appeared. This paper gives a design of the embedded SSL protocol stack. Furthermore, one kind of embedded network architecture model, which can be commonly used, is presented.
【Key words】 Embedded application; SSL protocol stack; Communication security

SSL本质上是一个通信协议，为了保证安全性，它的协议描述比较复杂，具有较完备的握手过程，这决定了它不是一个轻量级的网络协议。另外，SSL还涉及到大量的计算密集型算法：非对称加密算法，对称加密算法和数据摘要算法，虽然嵌入式平台的计算性能已经今非昔比，但是在运行这类算法时仍旧显得十分吃力。目前市场上存在为数不少集成SSL的产品（有免费的，也有商业性质的），它们都具有共同的特点：实现了SSL的全集，但是过分依赖于计算平台的强大计算能力；功能齐全，但是以牺牲可裁剪性为代价，未能实现SSL产品的独立发布。由于J2ME的广泛部署，J2ME也对其Java的实现增加了通过SSL提供安全传输支持的特性，但是这一特性被单独封装在J2ME的各种产品之中，无法单独被用户使用；同样的问题也存在于嵌入式网络产品特别是SSL-VPN网关中。

因此，无论是市场的需要还是技术的发展，都迫切需要一个能在嵌入式计算环境下运行稳定、具备良好可移植性和可剪裁性的SSL软件产品。

1 SSL协议栈设计

嵌入式SSL软件的设计目标是：具有通用性、能在嵌入式计算环境下良好地运行。因此在设计实现时，代码要尽可能紧凑，以适应嵌入式环境下的空间限制；其次，软件对计算能力的要求不能太高，并且能够针对特定的嵌入式环境进行优化，以适应嵌入式环境下的性能和时间限制。同时，由于SSL中包含的大量证书管理和算法在其他的安全应用(如数字版权管理DRM)中也被广泛使用，因此，应该把软件设计成一个协议栈和算法、证书管理库分离的结构，以便最大程度地支持软件复用。其中，SSL协议栈模块包含握手过程的建立和用户数据的封装和解封，并为用户提供编程接口；数字证书模块包含所有对数字证书的处理以及证书库的管理；算法模块包含所有涉及到的算法。SSL协议栈模块利用数字证书来完成对服务器端的认证，并利用算法模块提供的算法完成密钥交换、用户数据加/解密，利用单向散列函数完成MAC(消息完整性检查)。

1.1 体系结构设计

根据协议数据处理的流程和协议功能的相关性，将协议栈划分为4个模块，如图1所示。

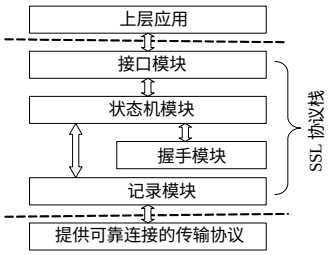

图1 SSL协议栈体系结构

其中，接口模块为上层应用提供调用接口；状态机模块处理协议模块的状态转换、驱动协议的运行；握手模块处理握手信息；记录层模块完成SSL数据包的封装/拆分、数据的加/解密。

1.2 状态机设计

建立新的安全连接的握手过程如图2所示。

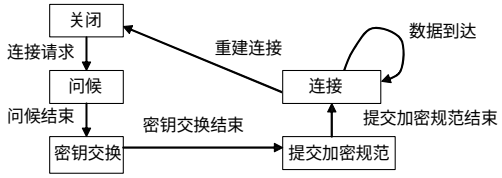

图2 建立新安全连接的握手过程

作者简介：罗建超(1978—)，男，助教、硕士，主研方向：嵌入式系统；俸 皓，讲师、硕士；罗 蕾，教授
收稿日期：2006-03-30 **E-mail：**andyluomail@163.com

重用已存在安全连接的握手过程即简化握手过程如图 3 所示。

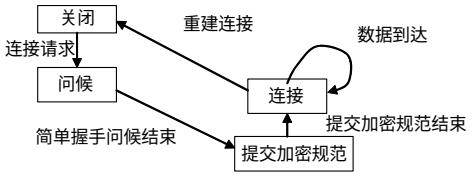

图 3 重用已存在安全连接的握手过程

对状态转换过程进行抽象，得到协议中的状态如下：

```
NULL,                          /* 空状态*/
CREATING,                      /* 会话创建状态*/
SERVER_HELLO,                  /* 接收 server hello 报文状态*/
SERVER_CERT,                   /* 认证 Server 证书状态*/
CLIENT_CERT,                   /* 认证 Client 证书状态*/
GET_CERT_REQ_OR_SERVER_HELLO_DONE,
/*接收 certificate request 报文或 server hello done 报文状态 */
SERVER_HELLO_DONE,             /* 接收 server hello done 报文状态 */
SEND_CLIENT_CERT,              /* 发送客户端证书状态*/
SEND_KEY_EXCHANGE,             /* 交换加密套件状态*/
SEND_CERT_VERIFY,              /* 发送客户端证书签名验证状态*/
SEND_CHANGE_CIPHER,            /* 改变安全参数状态*/
SEND_FINISH,                   /* 握手结束状态*/
GET_CHANGE_CIPHER,             /* 接收 change cipher spec 报文状态*/
GET_FINISH,                    /* 接收 finish 报文状态 */
OPENED                         /* 安全连接打开状态*/
```

状态机的状态转换如图 4 所示。

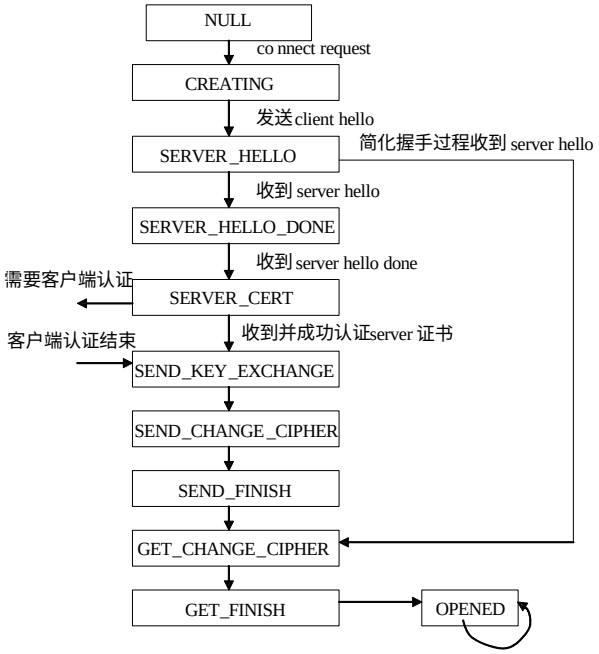

图 4 SSL 状态机状态转换

值得注意的是，在图中任何一个状态下，由于协商不成功或者超时导致状态转换正常推进失败，都将使得状态转换为 NULL 状态。如果收到上层发来的发送数据请求，而会话未处于 OPENED 状态，则转入到 CREATING 状态，为本次数据发送开始新的协商，建立新的安全连接进行发送。

1.3 运行设计

用户可使用协议栈向上提供的接口来进行一次安全数据操作，流程如图 5 所示。

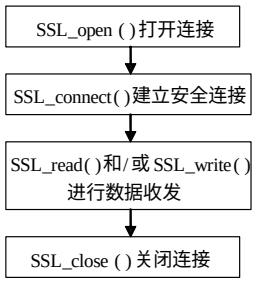

图 5 SSL 协议栈运行流程

若本次连接设定为可唤醒的，则在调用 SSL_close 时，只关闭本次连接，并不完全释放所有缓冲区，仍保留本次连接的参数。因此用户仍可使用本次协商之后的参数来建立下一连接，方法是不调用 SSL_open，而直接使用后续流程来操作。如果用户希望完全关闭，则可将连接设定为不可唤醒的，那么在调用 SSL_close 时就会把所有缓冲区和参数进行释放。

1.3.1 初始化过程设计

上层应用初始化协议栈。初始化记录协议状态的结构，将已打开/已关闭会话数清零，初始化协议支持算法链表，创建 SSL 任务。

1.3.2 安全连接建立过程设计

安全连接建立的过程即 SSL 的握手过程。该过程由对 SSL_open 的调用开始。在 SSL_open 过程中，首先新建会话并对其进行初始化：分配接收/发送缓冲，把会话插入会话链表，初始化会话的状态结构，将当前状态置为 NULL。接着，上层应用调用 SSL_connect，向状态机发送连接请求，然后调用状态机模块开始建立连接的过程。

1.3.3 数据发送过程设计

上层应用要发送数据时调用接口模块的 SSL_write 函数，SSL_write 函数把发送请求和数据交给状态机，状态机根据请求和当前状态决定下一步动作。如果状态机处于 OPENED 状态，则利用当前安全连接进行发送；如果状态机处于关闭（NULL）状态，则在建立一个安全连接后再发送。

1.3.4 数据接收过程设计

数据接收过程使用当前连接不断地从下层可靠传输协议获取数据，收到数据后，根据接收到的数据类型进行不同的处理：

(1)收到握手报文 将握手报文插入到接收握手报文的队列尾部，等待各个握手报文获取函数使用。握手报文获取函数先从握手报文消息队列内取下一个报文，如果是组合过的报文中的非最后一个报文，则根据消息长度判断取得一个所需报文，再把剩下的报文插回消息队列的头部，等待下一次调用握手报文获取函数的时候取用；如果该报文是组合报文的最后一个报文或者是单独的报文，则直接把经过解密等一系列处理获得的明文返回给调用者。

(2)收到应用数据报文：将收到的报文放到接收缓冲中，等待用户调用 SSL_read 进行读取。

(3)收到 change cipher spec 报文：由于协议规定此类报文具有较高的优先级，因此收到后立即初始化本次会话的用于解密的算法控制数据结构，准备开始密文传送。

(5)收到 alert 报文：因为协议中规定该类报文拥有最高的优先级，需要收到后立即处理，又由于在应用数据传输阶段出现的警告信息一般为密文，需要进行解密等一系列操作才可以得到警告信息的内容，为了复用已有的数据结构和处理方法，可先把 alert 报文插入握手报文消息队列的头部，再调用握手报文获取函数从队列中取得消息的明文，根据明文的内容再进行相应的处理。

1.3.5 安全连接关闭过程设计

连接的关闭有主动和被动之分。

(1)主动关闭连接的动作是由上层应用的函数调用发起的，流程为：上层调用函数 SSL_close，SSL_close 向状态机输入关闭连接请求，状态机收到该请求后，如果是可重用的会话，则发送 close notify 报文，关闭本次 SSL 会话，把状态机状态置为 NULL，退出；如果是不可重用的会话，则首先发送 close notify 报文，关闭下层可靠连接，把状态机状态置为 NULL，然后清除会话，释放会话结构的内存空间。

(2)被动关闭连接是在接收到 close notify 报文的警告或者致命警告后，下层连接被关闭后协议模块采取的动作。流程为：向对方发送 close notify 请求，不等待对方回应直接关闭连接，销毁本次会话。

1.3.6 会话管理设计

(1)会话管理：一个会话实体对应一个会话结构。综合考虑嵌入式应用的环境和需求，我们将会话和连接的对应关系简化为一一对应，即一个会话对应一个连接。会话结构被组织成单向链表的形式，其创建、插入、删除和维护由统一的链表操作来完成。会话的生存时间由会话数据结构上的 timeout 成员决定。

(2)会话缓冲管理：会话中的接收/发送缓冲按照环状缓冲区来组织。缓冲区按照配置给定的缓冲区大小进行初始化。每个缓冲区都有各自独立的读/写指针，当缓冲区满时，接收/发送操作立即返回错误信息。

2 一种通用嵌入式网络体系结构

与计算机总线上的通信协议一样，网络协议也分为两类：一类是面向数据传输的，如 PPP、PPTP、TCP、FTP 等；另一类是面向命令处理和控制的，如 Telnet、R-login 等。下面主要讨论面向数据传输的网络协议。

面向数据传输的网络协议，在不考虑广播方式的时候，大多具有如下的运行过程：握手→建立通信连接→数据发送/接收→关闭连接。握手过程主要是面向身份验证和通信参数的协商，有的协议甚至没有握手过程，只需要知道对端地址就可传输数据，通信的控制在传输过程中进行。各个协议的握手过程繁简不一，但无外乎一系列状态转换的抽象。

网络协议的目标系统在嵌入式环境下有较大的差异：有的是单任务系统，有的是多任务系统。在这种情况下，权衡了性能和实现的难易程度之后，将网络任务抽象成一个不断轮询的函数 XXX_check( )（XXX 在这里代表相应的网络协议名称）。在单任务环境下，它被加入到系统的主循环中；在多任务环境下，它按照一定的优先级被创建为一个伺服进程被系统使用。根据协议的功能，可以大致确定 XXX_check( )函数的工作：收/发数据报文并检查状态机。其中收发报文的操作需要参考相关的协议规范来具体实现，这里抽象为两个操作：XXX_pkt_demux()和 XXX_pkt_send()。其中 XXX_pkt_demux()负责从协议的服务层即下层协议接收(可以通过中断方式也可以通过轮询方式)并解析报文，而 XXX_pkt_send()负责发送本层协议的报文至下层协议。在分析了协议的状态转换之后，可以使用一组状态常量抽象表示协议的状态。状态的转换实现为 XXX_fsm_check()函数。因此，我们总结出协议核心操作 XXX_check()的内容如下：

```
XXX_check(void)
{ XXX_pkt_demux( );          /*接收报文*/
  XXX_fsm_check( );          /*检查状态机，转换状态*/
  XXX_pkt_send( );           /*发送报文*/ }
```

其中，对这 3 个核心操作的顺序进行了有意的编排：在检查状态机状态之前调用接收报文操作，这是因为协议状态的变迁往往是由于收到指定的报文、收到相应的事件引起的，因此，这样的顺序可以有效地避免状态机的空转，从而能提高系统效率。协议软件最重要的一部分是缓冲区的管理。下面给出一个通用的适合嵌入式系统的缓冲区管理部件模型，为简化起见，只给出接收缓冲的模型，发送缓冲同理。该模型如图 6 所示。

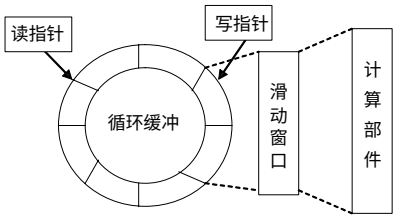

图 6 网络协议软件的接收缓冲模型

图 6 中可以看出，缓冲区被组织成循环缓冲的形式，这是为了便于管理和节省存储空间。循环缓冲的参考实现可以在连续的缓冲区之上(比如使用数组建立)，也可以由各个事先分配好的报文通过指针挂接到循环队列上实现。具体实现方案的选用，需要根据应用的场合来进行选择。滑动窗口是可选部件，在使用了滑动窗口技术的协议中，可以在缓冲区之上加入这一层。计算部件也是可选的，它负责对接收到的数据进行事先协商好的处理(比如解密、MAC 验证等等)。在我们的 SSL 协议设计中，由于保序、拥塞控制的工作都由下层可靠连接完成，因此没有滑动窗口这一层；另一方面，由于涉及到了加解密操作，因此我们在缓冲区之上拥有一套相应的计算部件。

考虑协议模型对多连接的支持，协议的连接可以被抽象成一组特定的数据结构。几个这样的数据结构的集合在系统里表征了多个连接的存在。这些数据结构可以被组织成一个队列，当这个队列作为输入被 XXX_check()依次处理的时候，就意味着我们的协议模型提供了对多连接的支持，这是系统中没有优先级的简单情况。如果存在多个优先级，那么可以按照优先级组织多个队列，并以优先级作为输入 XXX_check()的顺序的判断依据，在接收时同样需要小心地安排对不同优先级报文处理的顺序。

另外，在嵌入式网络协议中，参数的使用和保存也是值得注意的问题。一般来说，按照可以理解的数据结构组织好需要保存的参数，再使用专用的接口在可用的存储媒体(如 FLASH、SD 卡、MMC 卡等)上进行读写操作，进行参数的读取和保存。

3 结束语

本文给出了一种嵌入式 SSL 协议栈的设计方案，并进而提出了一种通用的嵌入式网络体系结构模型，具有较强的实用性和工程参考价值。该设计方案已被产品化，并在国内外多个移动终端平台上得到应用，经过实际网络环境的运行和测试，能够与多个应用服务提供商的网关建立安全连接并实现安全的数据通信。

参考文献

1 Transport Layer Security Working Group. The SSL Protocol(Version 3.0)[Z]. 1996-11-18.

2 TLS Protocol (Version 1.0)[S]. RFC2246, 1999.

3 Wagner D, Schneier B. Analysis of the SSL 3.0 Protocol[C]. Proc. of the 2nd Unix Workshop on Electronic Commerce, 1996.

4 Wolf W. 孙玉芳译. 嵌入式计算系统设计原理[M]. 北京: 机械工业出版社, 2002-02.