用 OpenSSL 进行 TLS/SSL 编程

范恒英 何大可
西南交通大学计算机与通信工程学院，成都 610031）

【摘 要】TLS/SSL 是一个得到广泛应用的安全通信标准，OpenSSL 是 TLS/SSL 的 Open Source 实现。介绍了 OpenSSL 提供的 TLS/SSL 编程接口，并给出了 WIN32 平台上用 OpenSSL 进行 TLS/SSL 编程的程序示例。

【关键词】传输层安全 安全套接层 OpenSSL

TLS/SSL Programming with OpenSSL

Fan Hengying He Dake
(Computer and Communication Engineering Inst., Southwest Jiaotong University, Chengdu 610031)

【Abstract】TLS/SSL is a widely – used standard for communication security, OpenSSL is an Open Source implementation of TLS/SSL. This paper introduces OpenSSL' s TLS/SSL programming interface and gives an OpenSSL – based TLS/SSL program example upon WIN32 platform.

【Keywords】TLS, SSL, OpenSSL

SSL(Secure Socket Layer) 是 Netscape 公司提出的安全通信协议。SSL3. 0[1]得到了业界广泛的认可，已成为事实上的标准。TLS(Transport Layer Security) 是 IETF 的 TLS 工作组在 SSL3. 0 基础之上提出的，目前版本是 1. 0。TLS1. 0[2] (可看作 SSL3. 1) 和 SSL3. 0 的差别不大，且考虑了和 SSL3. 0 的兼容性。TLS/SSL 可提供身份认证、机密性和完整性服务。OpenSSL[3] 是继承于 SSLeay0. 9. 0[4] 的 Open Source 软件，当前版本是 OpenSSL0. 9. 6c。OpenSSL 实现了 SSL2. 0、SSL3. 0 和 TLS1. 0。目前，OpenSSL 被众多的软件开发人员研究和使用。

1 OpenSSL 的 TLS/SSL 编程接口

使用 OpenSSL 的 ssl 库的典型 TLS/SSL 应用包含如下步骤 初始化 OpenSSL 的 ssl 库、建立 TLS/SSL 环境、建立 TLS/SSL 连接、使用 TLS/SSL 连接完成应用数据交换、关闭 TLS/SSL 连接并释放相关资源。

1. 1 初始化 OpenSSL 的 ssl 库

使用 SSL_library_init(void) 初始化 ssl 库，其作用是注册可用的密码和消息摘要算法。

1. 2 建立 TLS/SSL 环境

使用函数 SSL_CTX * SSL_CTX_new(SSL_METHOD * method) 来创建 SSL_CTX 结构，参数 method 是所用的连接方法。下面四组函数用于创建各种 SSL_MEHTOD，它们都没有参数，都返回 SSL_METHOD 结构的指针：

- SSLv2_method( ), SSLv2_server_method( ), SSLv2_client_method()。

使用这些函数返回的 SSL_METHOD 结构建立的 SSL 结构 (代表 SSL 连接) 只能理解 SSLv2 协议。SSLv2_server_method(void) 用于服务器端，SSLv2_client_method(void) 用于客户端，SSLv2_method(void) 用于既是客户又是服务器的情况，此时，通过在连接前对 SSL 结构使用 void SSL_set_connect_state(SSL * ssl) 或 void SSL_set_accept_state(SSL * ssl) 来指定是作为客户还是服务器。

- SSLv3_method( ), SSLv3_server_method( ), SSLv3_client_method()。

收稿日期：2002 – 02 – 27。
范恒英：男，1970 年生，硕士生。主要研究方向为网络通信安全与保密。
何大可：男，1944 年生，教授，博士生导师。主要研究方向为密码学、移动通信安全、并行机应用。

功能与属性与上一组函数类似，但相应的 SSL 结构只能理解 SSLv3 协议。

- TLSv1_method()，TLSv1_server_method()，TLSv1_client_method()。

功能与属性与第一组函数类似，但相应的 SSL 结构只能理解 TLSv1 协议。

- SSLv23_method()，SSLv23_server_method()，SSLv23_client_method()。

相应的 SSL 结构能理解 SSLv2、SSLv3、TLSv1 协议。可以使用 SSL_CTX_set_options()或 SSL_set_options()并指定 SSL_OP_NO_SSLv2、SSL_OP_NO_SSLv3、SSL_OP_NO_TLSv1 等选项来限定所使用的协议。

创建了 SSL_CTX 结构以后，可根据自己的需要进行设置，这里介绍几个常用的设置，其他的设置方法请参考 OpenSSL 的 man 页。

void SSL_CTX_set_verify(SSL_CTX * ctx, int mode, int (* verify_callback) (int, 509_STORE_CTX *)) 用于设置验证方式。mode 是以下值的逻辑或：SSL_VERIFY_NONE 表示不验证；SSL_VERIFY_PEER 用于客户端时要求服务器必须提供证书，用于服务器时服务器会发出证书请求消息要求客户端提供证书（但客户端也可不提供）；SSL_VERIFY_FAIL_IF_NO_PEER_CERT 只适用于服务器且必须与 SSL_VERIFY_PEER 一起使用，要求客户端必须提供证书。verify_callback 是处理验证的回调函数，如果没有特殊的需要，传入 NULL 表示使用 OpenSSL 提供的缺省验证函数。

int SSL_CTX_load_verify_locations(SSL_CTX * ctx, char * CAfile, char * CApath) 用于加载受信任的 CA 证书。CAfile 如果不为 NULL，则它指向的文件包含 PEM 编码格式的一个或多个 CA 证书。CApath 如果不为 NULL，则它指向一个包含 PEM 格式的 CA 证书的目录，目录中每一个文件包含一份 CA 证书，文件名是证书中 CA 名的 HASH 值，可使用 OpenSSL 软件包中提供的工具程序 c_rehash 参见 OpwnSSL 的 man 页）来建立该目录。SSL_CTX_load_verify_locations()返回 1 表示成功，返回 0 表示失败。

int SSL_CTX_use_certificate_file(SSL_CTX * ctx, const char * file, int type) 用于加载自己的证书。file 指向证书文件，type 指定证书文件的编码类型：SSL_FILETYPE_PEM 或 SSL_FILETYPE_ASN1。返回 1 表示成功，返回 0 表示失败。

int SSL_CTX_use_PrivateKey_file(SSL_CTX * ctx, const char * file, int type) 用于加载自己的私钥。file 指向私钥文件，type 指定私钥文件的编码类型：SSL_FILETYPE_PEM 或 SSL_FILETYPE_ASN1。返回 1 表示成功，返回 0 表示失败。

int SSL_CTX_check_private_key(SSL_CTX * ctx)验证私钥和证书是否相符。返回 1 表示相符，返回 0 表示不相符。

int SSL_CTX_set_cipher_list(SSL_CTX * ctx, const char * str) 设置允许使用的密码组，成功返回 1，失败返回 0。str 表示允许使用的密码组列表，列表的各部分以冒号分隔，如："RC4 - SHA: DES - CBC3 - SHA"，表示可使用密码组 TLS_RSA_WITH_RC4_ 128_SHA 和 TLS_RSA_WITH_3DES_EDE_CBC_SHA。OpenSSL 支持的密码组请参考 OpenSSL 的 man 页。

long SSL_CTX_set_mode(SSL_CTX * ctx, long mode)改变 SSL 输入/输出函数的工作模式。若原有的模式为 oldmode，则新的模式为 oldmode|mode，这里符号"|"表示'按位与"运算。要使阻塞模式下的 I/O 操作不要求重试，应设置模式 SSL_MODE_AUTO_RETRY。其他可用的模式请参考 OpenSSL 的 man 页。

1.3 建立 TLS/SSL 连接

完成 SSL 环境建立（由 SSL_CTX 结构表示）后，可以基于此环境建立多个 TLS/SSL 连接。一个 SSL 结构代表一个 TLS/SSL 连接，其中保存了该 TLS/SSL 连接需要的数据。

函数 SSL * SSL_new(SSL_CTX * ctx) 创建新的 SSL 结构。新建的 SSL 结构从 SSL 环境 ctx 继承设置。可以对 SSL 重新进行设置以取代从 SSL 环境中继承的缺省设置，新作的设置只对该 SSL 结构有效，与从同一环境创建的其他 SSL 结构无关。必须为每一个 SSL 结构设置底层的通信通道。下面仅介绍通信通道的设置方法，其他设置方法请参考 OpenSSL 的 man 页。

int SSL_set_fd(SSL * ssl, int fd) 将文件描述符 fd 作为 ssl 的输入输出设备（即通信通道），所有随后的数据 I/O 都将通过 fd 进行。通常，fd 是一个代表已经建立的套接字连接的套接字描述符。ssl 继承 fd 的行为特性，即如果 fd 是非阻塞的，那么 ssl 也是非阻

塞的;如果 fd 是阻塞的(缺省),则 ssl 也是阻塞的。SSL_set_fd()返回 1 表示成功,返回 0 表示失败。

设置好底层的通信通道后,可以显式的建立 TLS/SSL 连接,也可以直接进行读写操作,由读写操作去建立 TLS/SSL 连接。要显式建立连接,服务器端调用 int SSL_accept(SSL * ssl),客户端调用 int SSL_connect (SSL * ssl)。这两个函数返回 1 表示成功建立了 TLS/SSL 连接。返回 0 或 -1 表示失败或需要重试。应使用 int SSL_get_error(SSL * ssl, int ret) 确定是否需要重试,其中 ret 为 SSL_accept() 或 SSL_connect() 的返回值,若需要重试,则 SSL_get_error() 返回 SSL_ERROR_WANT_READ 或 SSL_ERROR_WANT_WRITE。

1.4 使用 TLS/SSL 连接完成应用数据交换

int SSL_read(SSL * ssl, char * buf, int num)从 ssl 读取最多 num 字节的数据到 buf 中。int SSL_write(SSL * ssl, char * buf, int num) 将 buf 中的 num 字节数据写入 ssl。这两个函数成功时返回实际读写的字节数。若返回 0 或负数,则表示调用失败或需要重试。若需要重试,SSL_get_error() 调用将返回SSL_ERROR_WANT_READ 或 SSL_ERROR_WANT_WRITE。

如果 TLS/SSL 连接尚未建立,则 SSL_read() 或 SSL_write()调用将自动建立 TLS/SSL 连接。如果连接的对方要求重新协商,SSL_read() 或 SSL_write() 调用也会自动处理。在阻塞模式下,SSL_read()或 SSL_write()调用直到成功执行或发生错误时才返回,除非需要重新协商连接参数,此时 SSL_read()或 SSL_write()调用也会立即返回并产生 SSL_ERROR_WANT_READ 或 SSL_ERROR_WANT_ WRITE,表示需要重试。要使阻塞模式下的 SSL_read() 或 SSL_write() 调用自动处理重试,可使用 SSL_MODE_AUTO_RETRY 标志调用 SSL_CTX_set_ mode()或 SSL_set_mode()。

1.5 关闭连接并释放相关资源

int SSL_shutdown(SSL * ssl) 关闭 TLS/SSL 连接。void SSL_free (SSL * ssl) 释放 SSL 结构。void SSL_CTX_free (SSL_CTX * ctx) 释放 SSL_CTX 结构。

2 示例程序(WIN32 版本)

以下的 TLS/SSL 客户端示例程序从 web 服务器取得 web 页并送到标准输出。程序使用阻塞模式(缺省)并设置了 SSL_MODE_AUTO_RETRY 标志以避免重试处理。有关证书文件的生成见参考文献[5]。程序编译时需指定连接 ssleay32.lib、libeay32.lib 和 ws2_32.lib,运行时需要动态连接库 libeay32.dll 和 ssleay32.dll。以上库文件在编译 OpenSSL 软件包后可在 out32dll 目录下找到(OpenSSL 软件包的编译及安装见 OpenSSL 软件包的说明文档)。

```
#include  <stdio.h>
#include  <stdlib.h>
#include  <winsock2.h>
#include  <openssl\err.h>
#include  <openssl\ssl.h>
int main()
{
    int ret; char tmpbuf[1024]; int sd;
    struct sockaddr_in sa; SSL_CTX * ctx;
    SSL * ssl; WSADATA wsaData;
    if(WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) {
        printf("WSAStartup()fail: %d\n", GetLastError());
        return -1;
    }
    SSL_library_init(); /* 初始化,参见 1.1 */
    /* 建立 SSL 环境,参见 1.2 */
    ctx = SSL_CTX_new(SSLv23_client_method());
    SSL_CTX_set_verify(ctx, SSL_VERIFY_PEER, NULL);
    if (!SSL_CTX_load_verify_locations(ctx, "cacert.pem", NULL)
        || !SSL_CTX_use_certificate_file(ctx, "clientcert.pem",
            SSL_FILETYPE_PEM)
        || !SSL_CTX_use_PrivateKey_file(ctx, "clientkey.pem",
            SSL_FILETYPE_PEM)
        || !SSL_CTX_check_private_key(ctx)) {
        fprintf(stderr, "Error setting up SSL_CTX\n");
        return -1;
    }
    SSL_CTX_set_cipher_list(ctx, "RC4-SHA:DES-CBC3-SHA");
    SSL_CTX_set_mode(ctx, SSL_MODE_AUTO_RETRY);
    /* 建立 TCP 连接 */
    sd = socket(AF_INET, SOCK_STREAM, 0);
    memset(&sa, 0, sizeof(sa));
    sa.sin_family = AF_INET;
    sa.sin_addr.s_addr = inet_addr("127.0.0.1");
    sa.sin_port = htons(4433);
    ret = connect(sd, (struct sockaddr *)&sa, sizeof(sa));
    if(ret < 0) {
        fprintf(stderr, "Error connect().\n");
        return -1;
    }
    /* 建立 TLS/SSL 连接,参见 1.3 */
    ssl = SSL_new(ctx); SSL_set_fd(ssl, sd);
    if(SSL_connect(ssl) <= 0) {
        fprintf(stderr, "Error SSL_connect().\n");
        return -1;
    }
    /* 通过 TLS/SSL 连接完成应用数据交换,参见 1.4 */
    ret = SSL_write(ssl, "GET/HTTP/1.0\n\n", strlen("GET/
                                        HTTP/1.0\n\n"));
    if(ret <= 0) {
        fprintf(stderr, "Error SSL_write().\n");
        return -1;
    }
```

```
    for(;;) {
        ret = SSL_read(ssl, tmpbuf, 1024);
        if(ret <= 0) break;
        fwrite(tmpbuf, 1, ret, stdout);
    }
    /* 关闭连接并释放相关资源,参见 1.5 */
    SSL_shutdown(ssl); shutdown (sd, 2);
    SSL_free(ssl); SSL_CTX_free(ctx);
    return 0;
}
```

3 结束语

TLS/SSL 协议支持众多的密码体制，提供了身份认证机制，可保证网络通信的机密性和完整性。OpenSSL 是各种版本的 TLS/SSL 的一份完整实现，在 Internet 特别是 WWW）中被广泛使用。要进一步了解 TLS/SSL 协议以及 OpenSSL，可参考后面列出的参考文献。

参考文献

1 SSL 3.0 SPECIFICATION. http://home.netscape.com/eng/ssl3/

2 The TLS Protocol Version 1.0. http://www.ietf.org/rfc/rfc2246.txt

3 OpenSSL 源程序及文档. http://www.openssl.org

4 SSLeay 0.9.0 http://maga.di.unito.it/security/resources/mirrors/SSLeay/SSLeay090/index.html

5 SSLeay Certificate Cookbook. http://www.ultranet.com/~fhirsch/Papers/cook/ssl_cook.html

（上接第 32 页）

• 判决、解码和解扰

相位校正后的信号输入判决模块进行判决得到判决结果 $d_x(kT)+jd_y(kT)$，进而得到 Y0Y1Y2Q3Q4Q5Q6，对 Y0Y1Y2 解码得到 Q1Q2，由此可以获得加扰的比特流，再经过解扰得到信息比特流。

• 载波相位估计

这个模块利用 $r_x'(kT)+jr_y'(kT)$ 和 $d_x(kT)+jd_y(kT)$ 对本地载波和信号的调制载波的相位之差 φ 进行估计。载波相位的同步非常重要，但是方法也很多[2,4,5]，这里不详细描述。

• 均衡器自适应调整

由于信道一般都是时间缓变的，均衡器必须自适应地调整。这里调整 LMS 算法进行调整。LMS 的好处是计算量小，跟踪能力强。它的缺点是收敛速度较慢。

• 先导序列检测与处理

V.17 标准在发送正式报文之前先要发送同步序列，在发送报文的过程中也可以根据需要再次发送。序列分为四段：第一段交替发送符号 A 和 B，共 256 个符号；第二段包含 2976 个符号（首次同步时）或者 38 个符号（再同步时）。在此阶段按照一定顺序发送 A、B、C、D 这四个符号；第三段是 64 个符号的桥接信号；最后一段发送的是全部为 1 的比特流经过扰乱之后的调制信号，长度为 48 个符号宽度。

本模块充分利用了同步序列。在接收 AB 序列的时候对载波相位进行同步，估计最佳判决时刻并据此进行采样时刻调整。接收第二段序列时对均衡器进行训练使其收敛。接收桥接信号时对均衡器继续训练。第四段扰 1 序列可以用来检验前两段的效果，判断能否在当前码率下工作。

一般情况下，接收机在处理完同步序列后就可以稳定地工作。只需做微小的跟踪调整。

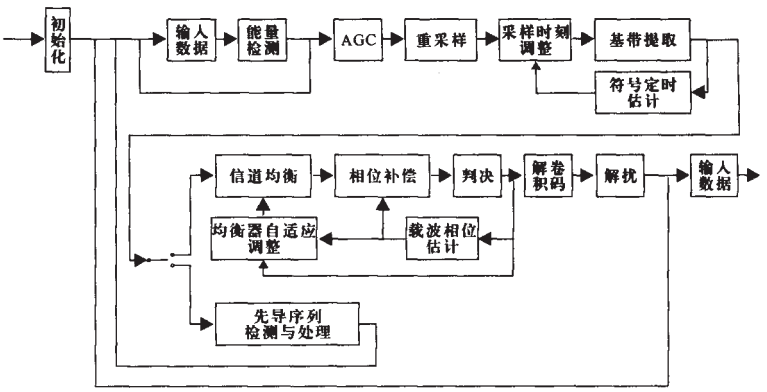

图 3 V.17 Soft modem 接收机功能结构图

5 结束语

首先分析了和传统调制解调器相比全软件实现的 Soft Modem 的特点和难点，随后介绍了一种用于传真的 V.17 Soft Modem 的实现方案。该方案已在 TMS320c6201 实时实现，计算量很低，只需要 4MCPS。用于 IP 电话网关等设备之中可以大大降低设备成本并增强系统的灵活性。

参考文献

1 ITU－T Recommendation V.17. Data communication over the telephone network a 2－wire modem for facsimile applications with rates up to 14400 bit/s. Geneva, 1991

2 Meyr H, Moeneclaey M, Fechtel S A. Digital Communication Receivers: Synchronization, Channel Estimation, and Signal Processing. New York: John Wiley & Sons, Inc. 1998

3 胡广书 编著. 数字信号处理. 北京：清华大学出版社，1997

4 Proakis J G 著，张力军、张宗橙、郑宝玉等译. 数字通信. 第 3 版，北京：电子工业出版社，2001

5 郭梯云等编著. 数据传输. 北京：人民邮电出版社，1998