A76xx Series Open SDK_SSL_应用指导

LTE 模组

芯讯通无线科技(上海)有限公司
上海市长宁区临虹路289号3号楼芯讯通总部大楼
电话：86-21-31575100
技术支持邮箱：support@simcom.com
官网：www.simcom.com

名称：	A76xx Series Open SDK_SSL_应用指导
版本：	V1.00
类别：	应用文档
状态：	已发布

版权声明

芯讯通无线科技(上海)有限公司
上海市长宁区临虹路289号3号楼芯讯通总部大楼
电话：86-21-31575100
邮箱：simcom@simcom.com
官网：www.simcom.com

了解更多资料，请点击以下链接：
http://cn.simcom.com/download/list-230-cn.html

技术支持，请点击以下链接：
http://cn.simcom.com/ask/index-cn.html 或发送邮件至 support@simcom.com

Version History

Version	Date	Owner	What is new
V1.00	2022-11-11		第一版

About this Document

本文档适用于 A1803S open 系列、A1603 open 系列、A1606 open 系列。

目录

缩略语

SSL	Secure Sockets Layer
TLS	Transport Layer Security

1SSL 简介

SSL(Secure Socket Layer)是 NetScape 公司提出的主要用于 web 的安全通信标准，分为 2.0 版和 3.0 版。TLS(Transport Layer Security)是 IETF 的 TLS 工作组在 SSL3.0 基础之上提出的安全通信标准。

1.1 SSL 协议介绍

SSL 是一个不依赖于平台和运用程序的协议，位于 TCP/IP 协议与各种应用层协议之间，为数据通信提高安全支持。下图是 SSL 在协议层中的位置。

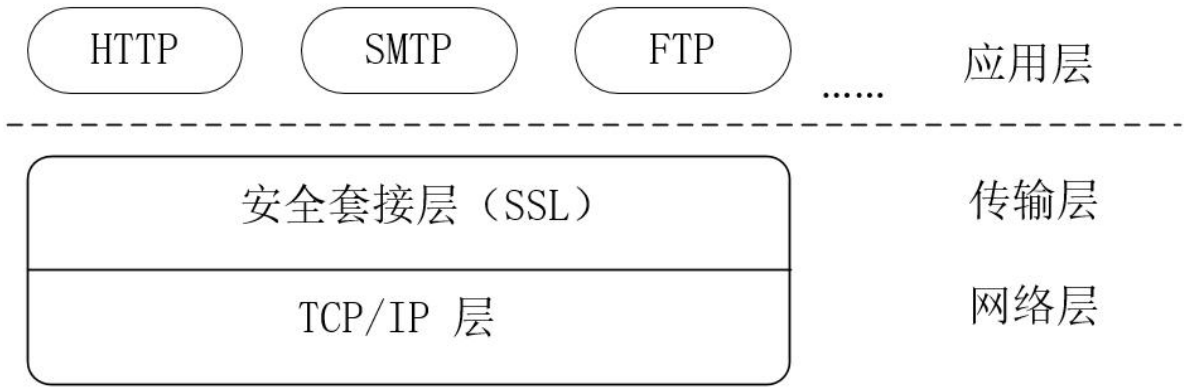


图 1.SSL 在协议层中的位置

1.2 SSL 协议结构

SSL 的体系结构可分为两个协议子层，其中底层是 SSL 记录协议层（SSL Record Protocol Layer）：它的作用是为高层协议提供基本的安全服务，SSL 纪录协议针对 HTTP 协议进行了特别的设计，使得超文本的传输协议 HTTP 能够在 SSL 运行，为高层协议提供数据封装、压缩解压缩、加密解密、计算和校验 MAC 等与安全有关的操作；高层是 SSL 握手协议层（SSL HandShake Protocol Layer）：SSL 握手协议层包括 SSL 握手协议（SSL HandShake Protocol）、SSL 密码参数修改协议（SSL Change Cipher Spec Protocol）和 SSL 告警协议（SSL Alert Protocol），握手层的这些协议用于 SSL 管理信息的交换，允许应用协议传送数据之间相互验证，协商加密算法和生成密钥等，其中 SSL 握手协议的作用是协调客户和服务器的状态，使双方能够达到状态的同步。

SSL 协议中最重要的是记录协议（SSL Record Protocol）和握手协议（SSL HandShake Protocol）：

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

SSL 握手协议：它建立在 SSL 记录协议之上，用于在实际的数据传输开始之前，通讯双方进行身份认证、协商加密算法、交换加密密钥等。

下图是 SSL 协议体系结构示意图。

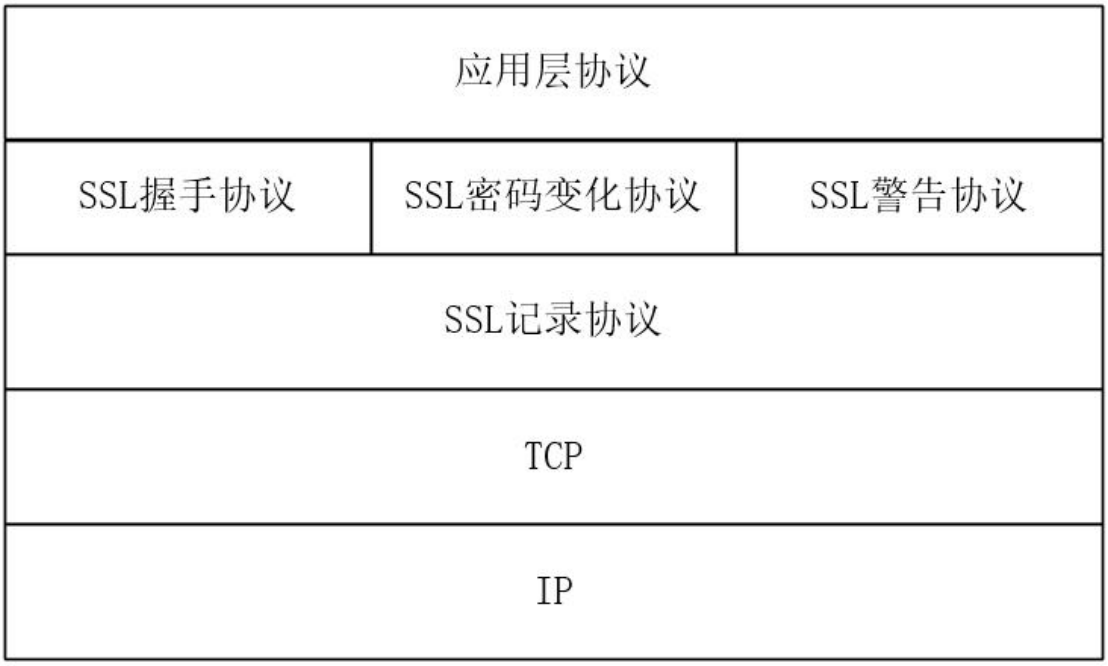
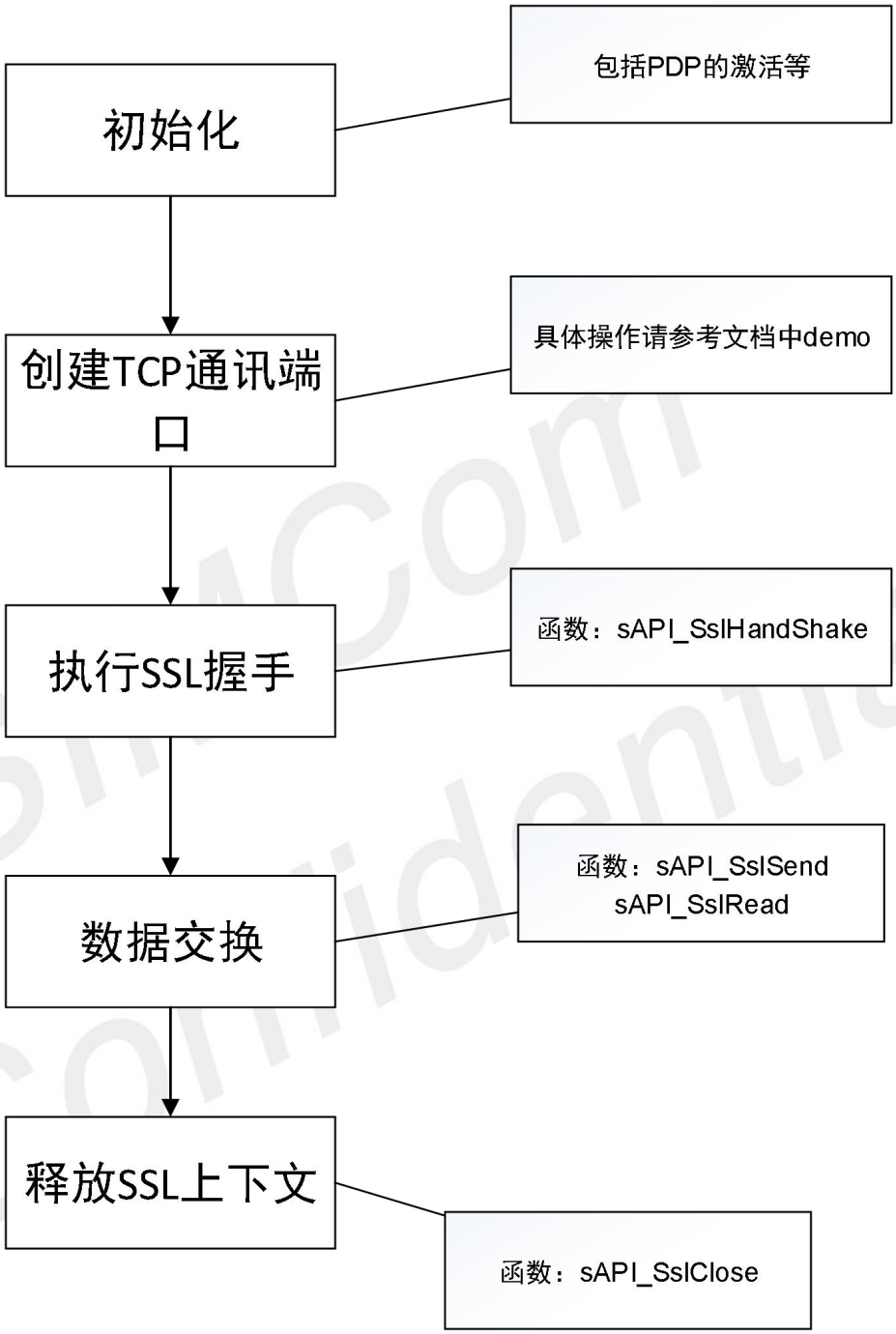


图 2.SSL 协议体系结构

1.3 使用流程



以上流程图只给出了简单的流程，详细操作请参考文档中 API 描述及 demo 内容。

2SSL API

SSL 头文件
#include "simcom_ssl.h"

2.1 sAPI_SsLGetContextIDMsg 获取上下文

获取 SSL 上下文配置

接口:	Int sslContent **sAPI_SsLGetContextIDMsg**(int sslId, SCsslContent *sslContext);
参数:	[in]**sslId**：SSL 上下文 ID。范围为 0-9。
返回值:	**sslContent**：包括 sslverson、authmode、ignoretime、negotiatetime、ca_file、clientcert_file、clientkey_file、enalbeSNI_flag、err。 **err**：0–获取内容成功 -1–获取内容失败。 **sslverson**：SSL 版本，默认值为 4。 0–SSL3.0 1–TLS1.0 2–TLS1.1 3–TLS1.2 4–all 服务器应支持配置的版本。因此，如果您不确定服务器支持的版本，则应使用默认值。 **authmode**：身份验证模式，默认值为 0。 0–无身份验证。 1–服务器身份验证。它需要服务器的根 CA。 2–服务器和客户端身份验证。它需要服务器的根 CA、客户端的证书和密钥。 3–客户端身份验证和无服务器身份验证。它需要客户端的证书和密钥。 **Ignoreltime**：指示如何处理过期证书的标志，默认值为 1。 0–关注认证的时间检查。 1–忽略认证时间检查。 当将值设置为 0 时，需要通过 AT+CCLK 在需要 SSL 认证时设置正确的当前日期和时间。 **ca_file**：SSL 上下文的根 ca 文件名。文件名的类型必须为“.pem”或“.der”。文件名的长度为 5 到 108 字节。 如果文件名包含非 ASCII 字符，则文件路径参数应包含前缀｛non-ASCII｝和引号（引号中的字符串应为文件名 UTF8 代码的十六进制）。 **clientcert_file**：与 ca_file 相同。 **clientkey_file**：与 ca_file 相同。 **enalbeSNI_flag**：指示是否启用 SNI 标志的标志，默认值为 0。 0–不启用 SNI。 1–启用 SNI。
NOTE:	

2.2 sAPI_SsLSetContextIDMsg 配置上下文

配置 SSL 上下文

接口:	int **sAPI_SsLSetContextIDMsg**(char* op, int sslId, char* value);
参数:	[in]**sslId**:SSL 上下文 ID。范围为 0-9。 [in]**op**:包括 sslverson、authmode、ignoretime、negotiatetime、ca_file、clientcert_file、clientkey_file、enalbeSNI_flag、err。 [in]**value**：配置 op 值，请参考 3.1 返回值。
返回值:	**0**：配置成功。 **-1**：配置失败。
NOTE:	

2.3 sAPI_SslHandShake 与服务器握手

与服务器进行握手

接口:	INT32 **sAPI_SslHandShake**(SCSslCtx_t *ctx);
参数:	[in]**ctx**：ssl 的上下文，请考虑结构体 SCSslCtx_t。
返回值:	**0**：握手成功。 **other**：握手失败。
NOTE:	

2.4 sAPI_SslRead 从服务器读取数据

从服务器读取数据

接口:	INT32 **sAPI_SslRead**(UINT32 ClientID, INT8 *buf, INT32 len);
参数:	[in]**ClientID**：ssl 的会话 id。 [in]**buf**：从网络接收的数据。 [in]**len**：你想读的长度。
返回值:	接收的字节数或非零错误代码。

NOTE:	

2.5 sAPI_SslSend 发送数据到服务器

将数据发送到服务器

接口:	INT32 **sAPI_SslSend**(UINT32 ClientID, INT8 *buf, INT32 len);
参数:	[in]**ClientID**：ssl 的会话 id。 [in]**buf**：从网络接收的数据。 [in]**len**：你想发送的长度。
返回值:	实际发送的字节数。
NOTE:	

2.6 sAPI_SslClose 释放上下文

释放 SSL 上下文

接口:	void **sAPI_SslClose**(UINT32 ClientID);
参数:	[in]**ClientID**：ssl 的会话 id
返回值:	无
NOTE:	

3 变量定义

3.1 SCSslCtx_t

```
typedef struct SCSslCtx_s {
    UINT32 ClientId;
    UINT32 ciphersuitesetflg;/* 0--使用默认密码套件，其他使用密码套件 */
    INT32   ciphersuite[8];
    /* ssl_version: 0---SLL3.0; 1---TLS1.0; 2----TLS1.1; 3----TLS1.2; 4----ALL; */
    UINT8   ssl_version;
    UINT8   enable_SNI;
    /*
    *  身份验证模式
    * 0：无认证
    * 1：管理服务器身份验证
    * 2：如果远程服务器请求，则管理服务器和客户端身份验证
    * 3：客户端身份验证和无服务器身份验证。它需要客户端的证书和密钥
    */
    UINT8   auth_mode;
    /* 0：关心认证的 tiem 检查；1：忽略 tiem 验证检查  */
    UINT8   ignore_local_time;
    /*  表示 SSL 协商阶段使用的最大超时，10-300 秒，默认值：300 秒  */
    INT8 *ipstr;

    INT8 *root_ca;
    UINT32 root_ca_len;
    INT8 *client_cert;
    UINT32 client_cert_len;
    INT8 *client_key;
    UINT32 client_key_len;
    /*带 PSK 的设备*/
    INT8 *psk;
    UINT32 psk_len;
    INT8 *psk_id;
    UINT32 psk_id_len;
    UINT32 fd;
    UINT32 timeout_ms;
    UINT16 negotiate_time;
```

```
}SCSslCtx_t;
```

3.2 SCsslContent

```
typedef struct{
    INT32 err;//0：成功  -1：失败
    UINT8  ssl_version;
    UINT8  ca_cert_path[64];
    UINT8  client_cert_path[64];
    UINT8  client_key_path[64];
    UINT8  enable_SNI;
    UINT8  auth_mode;
    UINT8  ignore_local_time;
    UINT16 negotiate_time;
}SCsslContent;
```

4Example

4.1 sAPI_SsLGetContextIDMsg

```
#include "simcom_ssl.h"
Int ssl_testMain(void);
{
    sslContent content = {0};
    content = sAPI_SsLGetContextIDMsg(0);
    if(0 == content.err);
    {
        ……    //get content success
    }
    else
    {
        return -1;
    }
}
```

4.2 sAPI_SsLSetContextIDMsg

```
#include "simcom_ssl.h"
Int ssl_testMain(void);
{
    int rest = 0;
    ret = sAPI_SsLSetContextIDMsg("sslversion", 0, 4);
    if(0 == ret);
    {
        ……    //configure successful.
    }
    else
    {
        return -1;
    }
}
```

4.3 sAPI_SslHandShake

```
#include "simcom_ssl.h"
void demo_ssl_test(void);
{
	INT32 ret = 0;
	SCSslCtx_t ctx;
	ctx.fd = sockfd;
	ctx.ssl_version = SC_SSL_CFG_VERSION_ALL;


	ret = sAPI_SslHandShake(&ctx);
	sAPI_Debug("sAPI_SslHandShake ret[%d] ", ret);

	if(ret == 0);
	{
		sAPI_Debug("sAPI_SslHandShake success");
	}
	else
	{
		sAPI_Debug("sAPI_SslHandShake fail");
	}
}
```

4.4 sAPI_SslRead

```
#include "simcom_ssl.h"
void demo_ssl_test(void);
{
	memset(recvbuf, 0x0, 1024);
	ret = sAPI_SslRead(0, recvbuf, 1024);
	sAPIebug("ret [%d] recvbuf[%s] ", ret, recvbuf);
	if(ret> 0);
	{
		sAPI_Debug("sAPI_ SslRead success");
	}
	else
	{
		sAPI_Debug("sAPI_ SslRead errono:%d", ret);
	}
}
```

4.5 sAPI_SslSend

```
#include "simcom_ssl.h"
void demo_ssl_test(void);
{
	ret = sAPI_SslSend(0, sendbuf, sizeof(sendbuf););
	sAPI_Debug("ret [%d] sendbuf[%s] ", ret, sendbuf);
	if(ret> 0);
	{
		sAPI_Debug("sAPI_ SslSend success");
	}
	else
	{
		sAPI_Debug("sAPI_ SslSend fail);
	}
}
```

4.6 sAPI_SslClose

```
#include "simcom_ssl.h"
void demo_ssl_test(void);
{
	sAPI_SslClose(0);
}
```

5 参考 demo

SDK 中提供的参考 demo：demo_ssl.c

```
#include "string.h"
#include "stdlib.h"
#include "stdio.h"
#include "simcom_ssl.h"
#include "simcom_tcpip.h"
#include "simcom_tcpip_old.h"
#include "simcom_debug.h"
#include "simcom_os.h"
#include "simcom_common.h"

extern sMsgQRef simcomUI_msgq;
extern void PrintfOptionMenu(char* options_list[], int array_size);
extern void PrintfResp(char* format);
extern void SslTestDemoInit(int sslTestTime);

/**
  * @brief   SSL Demo.
  * @param   void
  * @note
  * @retval void
  */
void SslDemo(void)
{
    INT32 ret = 0;
    SCSslCtx_t ctx;
    INT32 sockfd = -1;
    SChostent *host_entry = NULL;
    SCsockAddrIn server;
    UINT32 opt = 0;
    INT8 recvbuf[1024];
    INT8 sendbuf[] = { /* Packet 8 */
    0x47, 0x45, 0x54, 0x20, 0x2f, 0x20, 0x48, 0x54,
    0x54, 0x50, 0x2f, 0x31, 0x2e, 0x31, 0x0d, 0x0a,
    0x48, 0x6f, 0x73, 0x74, 0x3a, 0x20, 0x77, 0x77,
    0x77, 0x2e, 0x62, 0x61, 0x69, 0x64, 0x75, 0x2e,
    0x63, 0x6f, 0x6d, 0x0d, 0x0a, 0x43, 0x61, 0x63,
    0x68, 0x65, 0x2d, 0x43, 0x6f, 0x6e, 0x74, 0x72,
```

```
0x6f, 0x6c, 0x3a, 0x20, 0x6e, 0x6f, 0x2d, 0x63,
0x61, 0x63, 0x68, 0x65, 0x0d, 0x0a, 0x43, 0x6f,
0x6e, 0x74, 0x65, 0x6e, 0x74, 0x2d, 0x54, 0x79,
0x70, 0x65, 0x3a, 0x20, 0x74, 0x65, 0x78, 0x74,
0x2f, 0x70, 0x6c, 0x61, 0x69, 0x6e, 0x0d, 0x0a,
0x41, 0x63, 0x63, 0x65, 0x70, 0x74, 0x3a, 0x20,
0x2a, 0x2f, 0x2a, 0x0d, 0x0a, 0x0d, 0x0a, 0x0d,
0x0a };

SIM_MSG_T optionMsg ={0,0,0,NULL};
char *note = "\r\nPlease select an option to test from the items listed below.\r\n";
char *options_list[] = {
"1. SSL CONNECT DEMO",
"2. TEST FOR SSL",
"3. ",
"99. back",
 };
 /*{
"1. test for ssl",
"99. back",
 }; */

while(1)
{
    PrintfResp(note);
    PrintfOptionMenu(options_list,sizeof(options_list)/sizeof(options_list[0]));

    sAPI_MsgQRecv(simcomUI_msgq,&optionMsg,SC_SUSPEND);
    if(SRV_UART != optionMsg.msg_id)
    {
        sAPI_Debug("%s,msg_id is error!!",__func__);
        break;
    }

    sAPI_Debug("arg3 = [%s]",optionMsg.arg3);
    opt = atoi(optionMsg.arg3);
    free(optionMsg.arg3);
    switch(opt)
    {
        case SC_SSL_COMMUNICATE:                                    //选择 1
            {
                char path[1024 + 256] = {0};
                if (-1 == sAPI_TcpipPdpActive(1,1))
                {
                    sAPI_Debug("PDP active err");
```

```
        snprintf(path,sizeof(path),"\r\nPDP active err\r\n");
        PrintfResp(path);
        break;
    }

    sockfd = sAPI_TcpipSocket(SC_AF_INET, SC_SOCK_STREAM, 0);   //创建 socket

    sAPI_Debug("demo_ssl_test sockfd[%d]",sockfd);
    if(sockfd < 0)
    {
        sAPI_Debug("create socket err");
        snprintf(path,sizeof(path),"\r\ncreate socket err\r\n");
        PrintfResp(path);
        if (-1 == sAPI_TcpipPdpDeactive(1,1))
        {
            sAPI_Debug("PDP deactive err");
            snprintf(path,sizeof(path),"\r\nPDP deactive err\r\n");
            PrintfResp(path);
        }
        break;
    }

    host_entry = sAPI_TcpipGethostbyname((INT8 *)"www.baidu.com");   //获取 host
    if (host_entry == NULL)
    {
        sAPI_SslClose(0);
        sAPI_TcpipClose(sockfd);
        sAPI_Debug("DNS gethostbyname fail");
        snprintf(path,sizeof(path),"\r\nDNS gethostbyname fail\r\n");
        PrintfResp(path);
        if (-1 == sAPI_TcpipPdpDeactive(1,1))
        {
            sAPI_Debug("PDP deactive err");
            snprintf(path,sizeof(path),"\r\nPDP deactive err\r\n");
            PrintfResp(path);
        }
        break;

    }

    server.sin_family = SC_AF_INET;
    server.sin_port = sAPI_TcpipHtons(443);
    server.sin_addr.s_addr= *(UINT32 *)host_entry->h_addr_list[0];

    sAPI_Debug("start connect!!!");
```

```
ret = sAPI_TcpipConnect(sockfd,(SCsockAddr *)&server,sizeof(SCsockAddr));
if(ret != 0)
{
    sAPI_SslClose(0);
    sAPI_TcpipClose(sockfd);
    sAPI_Debug("connect server fail");
    snprintf(path,sizeof(path),"\r\nconnect server fail\r\n");
    PrintfResp(path);
    if (-1 == sAPI_TcpipPdpDeactive(1,1))
    {
        sAPI_Debug("PDP deactive err");
        snprintf(path,sizeof(path),"\r\nPDP deactive err\r\n");
        PrintfResp(path);
    }
    break;
}
memset(&ctx,0,sizeof(ctx));
ctx.fd = sockfd;
ctx.ssl_version = SC_SSL_CFG_VERSION_ALL;

ret = sAPI_SslHandShake(&ctx);                                  //执行 SSL 握手
sAPI_Debug("sAPI_SslHandShake ret[%d]",ret);

if(ret == 0)
{
    ret = sAPI_SslSend(0,sendbuf,sizeof(sendbuf));
    sAPI_Debug("ret [%d] sendbuf[%s]",ret,sendbuf);
    memset(recvbuf,0x0,1024);
    ret = sAPI_SslRead(0,recvbuf,1024);                         //数据交换
    sAPI_Debug("ret [%d] recvbuf[%s]",ret,recvbuf);
    sAPI_Debug("sApi_SslConnect [%d]",ret);
    sAPI_SslClose(0);                                           //释放 SSL 上下文
    sAPI_TcpipClose(sockfd);
    snprintf(path,sizeof(path),"\r\nret [%d] recvbuf[%s]\r\n",(int)ret,(char *)recvbuf);
    PrintfResp(path);

    if (-1 == sAPI_TcpipPdpDeactive(1,1))
    {
        sAPI_Debug("PDP deactive err");
        snprintf(path,sizeof(path),"\r\nPDP deactive err\r\n");
        PrintfResp(path);
    }
    break;

}
```

```
                else
                {
                    sAPI_SslClose(0);
                    sAPI_TcpipClose(sockfd);
                    snprintf(path,sizeof(path),"\r\nhandshake fail,ret:%d\r\n",(int)ret);
                    PrintfResp(path);
                    if (-1 == sAPI_TcpipPdpDeactive(1,1))
                    {
                        sAPI_Debug("PDP deactive err");
                        snprintf(path,sizeof(path),"\r\nPDP deactive err\r\n");
                        PrintfResp(path);
                    }
                    break;
                }
            }

        case SC_SSL_TEST:                                                    //选择 2
            {
                sAPI_Debug("Start SSL demo test thread!");
                int sslTestTime = 0;
                PrintfResp("\r\nPlease input test time.\r\n");
                //optionMsg = GetParamFromUart();
                SIM_MSG_T option_Msg ={0,0,0,NULL};
                sAPI_MsgQRecv(simcomUI_msgq,&option_Msg,SC_SUSPEND);
                sAPI_Debug("arg3 = [%s]",option_Msg.arg3);
                sslTestTime = atoi(option_Msg.arg3);
                SslTestDemoInit(sslTestTime);
                break;
            }

        case SC_SSL_DEMO_MAX:                                                //选择 99
            {
                return;
            }

        default :
             break;
    }

  }
}
```