XRADIO MBEDTLS

Developer Guide

Revision 1.0

Oct 23, 2019

Declaration

THIS DOCUMENTATION IS THE ORIGINAL WORK AND COPYRIGHTED PROPERTY OF XRADIO TECHNOLOGY (“XRADIO”). REPRODUCTION IN WHOLE OR IN PART MUST OBTAIN THE WRITTEN APPROVAL OF XRADIO AND GIVE CLEAR ACKNOWLEDGEMENT TO THE COPYRIGHT OWNER.

THE PURCHASED PRODUCTS, SERVICES AND FEATURES ARE STIPULATED BY THE CONTRACT MADE BETWEEN XRADIO AND THE CUSTOMER. PLEASE READ THE TERMS AND CONDITIONS OF THE CONTRACT AND RELEVANT INSTRUCTIONS CAREFULLY BEFORE USING, AND FOLLOW THE INSTRUCTIONS IN THIS DOCUMENTATION STRICTLY. XRADIO ASSUMES NO RESPONSIBILITY FOR THE CONSEQUENCES OF IMPROPER USE (INCLUDING BUT NOT LIMITED TO OVERVOLTAGE, OVERCLOCK, OR EXCESSIVE TEMPERATURE).

THE INFORMATION FURNISHED BY XRADIO IS PROVIDED JUST AS A REFERENCE OR TYPICAL APPLICATIONS, ALL STATEMENTS, INFORMATION, AND RECOMMENDATIONS IN THIS DOCUMENT DO NOT CONSTITUTE A WARRANTY OF ANY KIND, EXPRESS OR IMPLIED. XRADIO RESERVES THE RIGHT TO MAKE CHANGES IN CIRCUIT DESIGN AND/OR SPECIFICATIONS AT ANY TIME WITHOUT NOTICE.

NOR FOR ANY INFRINGEMENTS OF PATENTS OR OTHER RIGHTS OF THE THIRD PARTIES WHICH MAY RESULT FROM ITS USE. NO LICENSE IS GRANTED BY IMPLICATION OR OTHERWISE UNDER ANY PATENT OR PATENT RIGHTS OF XRADIO.THIRD PARTY LICENCES MAY BE REQUIRED TO IMPLEMENT THE SOLUTION/PRODUCT. CUSTOMERS SHALL BE SOLELY RESPONSIBLE TO OBTAIN ALL APPROPRIATELY REQUIRED THIRD PARTY LICENCES. XRADIO SHALL NOT BE LIABLE FOR ANY LICENCE FEE OR ROYALTY DUE IN RESPECT OF ANY REQUIRED THIRD PARTY LICENCE. XRADIO SHALL HAVE NO WARRANTY, INDEMNITY OR OTHER OBLIGATIONS WITH RESPECT TO MATTERS COVERED UNDER ANY REQUIRED THIRD PARTY LICENCE.

Revision History

Version	Date	Summary of Changes
1.0	2019-10-23	Initial Version

Contents

1 模块概要

1.1 功能介绍

Mbedtls 库支持一组加密组件，包括 SSL/TLS 通信模块、TCP/IP 通信模块、哈希模块、RNG 模块、Cipher 模块、PK 模块、X.509 模块，并且这些模块可以通过配置文件进行独立使用。SSL/TLS 模块提供整体安全框架；TCP/IP 模块提供下层网络通信接口；Hash 模块提供散列算法；RNG 模块提供随机数；Cipher 模块提供对称加密算法；X.509 提供证书相关接口。

SDK 同时移植了 mbedtls2.2.0 和 mbedtls2.16.0 版本，为了用户开发方便、快速搭建安全服务。移植过程中，在 mbedtls 接口库的基础上对证书加密通信接口做了进一步的封装，本文对封装后的接口做了详细的介绍。如果接口不满足读者需求，读者可以直接调用 mbedtls 提供的原始接口，源代码中也提供了大量的例程供参考。

1.2 代码位置

模块	文件类型	文件位置
MBEDTLS	source	sdk/src/net/mbedtls-2.2.0 sdk/src/net/mbedtls-2.16.0
	header	sdk/include/net/mbedtls
	demo	sdk/project/common/cmd/tls

2 模块结构体

本章中的模块结构体并不是 mbedtls 源码的结构体，而是经过封装的结构体。有些结构体或有些结构体里面的变量是给 mbedtls 内部使用的，用户不需要配置和操作。

2.1 mbedtls context 结构体

```
typedef struct {
    int is_client; //是客户端模式还是服务器模式，0：客户端，1：服务器
    crt_context cert; //证书和私钥结构体
    mbedtls_entropy_context entropy; //mbedtls 的熵上下文结构体，是提供给 mbedtls 内部使用
    mbedtls_ctr_drbg_context ctr_drbg; //mbedtls 随机数上下文结构体，是提供给 mbedtls 内部使用
    mbedtls_ssl_context ssl; //mbedtls 上下文结构体，是提供给 mbedtls 内部使用
    mbedtls_ssl_config conf; //mbedtls 配置上下文结构体，是提供给 mbedtls 内部使用
} mbedtls_context;
```

2.2 证书结构体

```
typedef union {
    struct {
        mbedtls_x509_crt ca; //客户端模式下，CA 证书结构体，是提供给 mbedtls 内部使用
        mbedtls_x509_crt cert; //客户端模式下，自己的证书，是提供给 mbedtls 内部使用
        mbedtls_pk_context key; //客户端模式下，自己的私钥，是提供给 mbedtls 内部使用
    } cli_cert;
    struct {
        mbedtls_x509_crt cert; //服务器模式下，证书链，是提供给 mbedtls 内部使用
        mbedtls_pk_context key; //服务器模式下，自己的私钥，是提供给 mbedtls 内部使用
    } srv_cert;
} crt_context;
```

2.3 服务器证书结构体

服务器模式下，需要三个文件：CA 证书，自己的证书，自己的私钥。CA 证书和自己的证书构成一个证书链。

```
typedef struct {
    char *pCa; //CA 证书
    unsigned int nCa; //CA 证书数据长度
    char *pCert; //自己的证书
    unsigned int nCert; //自己证书数据长度
    char *pKey; //自己的私钥
    unsigned int nKey; //自己私钥的长度
} security_server;
```

2.4 客户端证书结构体

客户端模式下，需要三个文件：CA 证书，自己的证书，自己的私钥。CA 证书用来验证服务器，自己的证书和私钥用于双相验证。下面结构体中，pCa 变量和 certs 中的 pCa 变量是一样的，都是 CA 证书，这样重复设计的原因是因为在绝大部分连接中，都是单向验证，不需要设置客户端证书（只有在双向验证中才需要配置 certs 变量），这样设计便于代码编写方便。

```
typedef struct {
    char *pCa; //CA 证书
    unsigned int nCa; //CA 证书数据长度
    security_server certs; //在双相验证中，该变量里面的证书和私钥是都客户端的，而不是服务器的。
} security_client;
```

3 模块接口

3.1 创建 tls socket

mbedtls_sock* mbedtls_socket(int nonblock)		备注
功能	创建 tls 类型的 socket。	
参数	nonblock：非阻塞标志。1：非阻塞；0：阻塞	
返回值	mbedtls socket 指针	

3.2 关闭 tls socket

int mbedtls_closesocket(mbedtls_sock* fd)		备注
功能	关闭 tls socket	
参数	fd：mbedtls socket 指针	
返回值	0：成功	

3.3 初始化 tls 上下文

mbedtls_context* mbedtls_init_context(int client)		备注
功能	创建 tls 类型的上下文，内部会申请 mbedtls 需要的资源	
参数	client：当前为客户端还是服务器，1：客户端，0：服务器	
返回值	mbedtls 上下文指针	

3.4 反初始化 mbedtls 上下文

void mbedtls_deinit_context(mbedtls_context *context)		备注
功能	释放 tls 上下文，释放内部资源	
参数	context：上下文指针	
返回值	void	

3.5 配置 tls 上下文

int mbedtls_config_context(mbedtls_context *context, void *param, int verify)		备注
功能	配置 tls 上下文，将客户端/服务器配置好的证书结构体配置到 mbedtls	
参数	context：tls 上下文指针 param：指向证书结构体（server/client）的指针 verify：证书校验模式	
返回值	0：成功，其他：失败	

3.6 连接服务器

int mbedtls_connect(mbedtls_context *context, mbedtls_sock* fd, struct sockaddr *name, int namelen, char *hostname)		备注
功能	连接远程服务器，这个只是 tcp 层面上的连接，tcp 连接后，还需要握手	
参数	context：tls 上下文指针 fd：mbedtls socket 上下文指针 name：服务器端的 sockaddr namelen：name 的长度 hostname：server 的域名	
返回值	0：成功，其他：失败	

3.7 mbedtls 握手

int mbedtls_handshake(mbedtls_context *context, mbedtls_sock* fd)		备注
功能	执行握手流程	
参数	context：tls 上下文指针 fd：mbedtls socket 上下文指针	
返回值	0：成功，其他：失败	

3.8 发送数据

int mbedtls_send(mbedtls_context *context, char *buf, int len)		备注
功能	发送数据	
参数	context：tls 上下文指针 buf：存储数据的 buffer 指针 len：buffer 的长度	
返回值	0：成功，其他：失败	

3.9 接收数据

int mbedtls_recv(mbedtls_context *context, char *buf, int len)		备注
功能	接收数据	
参数	context：tls 上下文指针 buf：发送的数据指针 len：数据长度	
返回值	0：成功，其他：失败	

3.10 查询剩余数据

int mbedtls_recv_pending(mbedtls_context *context)		备注
功能	用于查询 tls 层剩下未读取的数据。	
参数	context：tls 上下文指针	
返回值	剩余数据的长度	

3.11 接受连接

int mbedtls_accept(mbedtls_context *context, mbedtls_sock *local_fd, mbedtls_sock *remote_fd)		备注
功能	接受一个新的连接，用于服务器模式下	

参数	context：tls 上下文指针 local_fd：本地 mbedtls socket 上下文指针 remote_fd：远端 mbedtls socket 上下文指针	
返回值	0：成功，其他：失败	

4 模块接口例程

本节提供上节介绍接口的示例，描述接口的使用方法及流程，文中代码皆为参考代码，不能直接运行。shttpd 与 httpclient 模块的安全传输机制，都是基于上述接口的实现，读者可以参考 HTTPMbedTLSWrapper.c、io_ssl.c 的调用过程。

注意：本节不会描述 mbedtls 原生接口的使用，有兴趣的用户可以参考官方实例（sdk/src/net/mbedtls-2.2.0/programs/）。

4.1 调用流程

第一步：获取网络上下文

客户端：

```
//创建阻塞的网络上下文
mbedtls_sock*netfd = mbedtls_socket(0);
```

服务器：

```
//创建阻塞的网络上下文
mbedtls_sock remote_fd;
mbedtls_sock*local_fd = mbedtls_socket(0);
```

第二步：创建并初始化 tls 上下文

客户端：

```
//创建并初始化 tls 客户端上下文
mbedtls_context *pContext = (mbedtls_context *)mbedtls_init_context(0);
```

服务器：

```
//创建并初始化 tls 服务器上下文
mbedtls_context *pContext = (mbedtls_context *)mbedtls_init_context(1);
```

第三步： 配置 tls 上下文

客户端：

```
//配置 tls 客户端上下文
if ((ret = mbedtls_config_context(pContext, (void *) &client_param,
MBEDTLS_SSL_CLIENT_VERIFY_LEVEL)) != 0) {
    HC_ERR(("https: config failed.."));
    return -1;
}
```

服务器：

```
//配置 tls 客户端上下文
mbedtls_config_context(ssl_context,          (void      *)      &server_param,
MBEDTLS_SSL_SERVER_VERIFY_LEVEL);
```

注意：1.客户端与服务器所需要的证书参数是不同的，请参考具体代码。

2.客户端与服务器的证书验证模式通常是不同的。

第四步： 建立连接

客户端：

```
//连接服务器
if ((ret = mbedtls_connect(pContext, (mbedtls_sock*) &net_fd, ServerAddress,
namelen, hostname)) != 0) {
    HC_ERR(("https: connect failed.."));
    return -1;
}
```

服务器：

```
//等待客户端连接
ret = mbedtls_accept(mbedtls_context *context, mbedtls_sock *local_fd,
&remote_fd);
```

第五步：协商握手

```
//执行握手流程
if ((ret = mbedtls_handshake(g_pContext, &g_httpc_net_fd)) != 0)
    return -1;
```

第六步：发送/接收数据

```
//发送数据
if ((ret = mbedtls_send(g_pContext, buf, len)) < 0)
    return -1;

//接收数据
if ((ret = mbedtls_recv(g_pContext, buf, len)) < 0)
    return -1;
```

第七步：释放 tls 上下文

```
//释放 tls 上下文
mbedtls_deinit_context(g_pContext);
```

第八步：释放网络上下文

客户端：

```
//释放网络上下文
mbedtls_closesocket(net_fd);
```

服务器：

```
//释放网络上下文
mbedtls_closesocket(local_fd);
mbedtls_closesocket(remote_fd);
```

5 其他配置

更多配置选项可参考《AW1825_MBEDTS_Appication_Note.docx》文档。

1. 修改 Makefile 中配置头文件

修改方式：

mbedtls 源码目录中 makfile 文件中的“CC_FLAGS += -DMBEDTLS_CONFIG_FILE='<config-xr-mini-cliserv.h>'”。

config-xr-mini-cli.h　　//支持 client

config-xr-mini-cliserv.h　　//支持 client/server

config-xr-mini-serv.h　　//支持 server

若上述三种配置均不能满足用户的要求，需要重新生成配置头文件。

SDK 默认使用的是 config-xr-mini-cliserv.h 配置文件。

备注：绝大部分情况下可以不用修改

2. 调试打印接口（位于配置头文件中）

#define MBEDTLS_DEBUG_C 打开该宏

备注：该宏只能打开本文接口中的 mbedtls 调试，如果需要打开任意 mbedtls 连接的打印，可参考《AW1825_MBEDTS_Appication_Note.docx》文档

3. 配置输入输出 buffer 大小（位于配置头文件中）

#define MBEDTLS_SSL_MAX_CONTENT_LEN　　(16*1024)

备注：为了增加兼容性，默认使用 16K