ESP8266 SSL 加密

使用手册

版本 2.0

关于本手册

本文介绍基于 ESP8266_NONOS_SDK 的 SSL 加密使用方法。

章	标题	内容
第 1 章	概述	介绍 SSL 概况
第 2 章	环境搭建	如何搭建编译环境
第 3 章	ESP8266 作为 SSL Server	介绍 ESP8266 作为 SSL server 时的使用方法
第 4 章	ESP8266 作为 SSL Client	介绍 ESP8266 作为 SSL client 时的使用方法
第 5 章	软件接口	介绍 SSL 软件接口

发布说明

日期	版本	发布说明
2016.11	V1.0	首次发布
2018.01	V2.0	重大更新。

文档变更通知

用户可通过乐鑫官网订阅技术文档变更的电子邮件通知。

证书下载

用户可通过乐鑫官网下载产品证书。

目录

1. 概述

SSL 指安全套接层 (Secure Socket Layer)，而 TLS 是 SSL 的继任者，称为传输层安全 (Transport Layer Security)。SSL/TLS 位于 TCP/IP 协议与上层应用协议之间，为数据通讯提供安全支持。例如，HTTP 协议是明文传输，加上 SSL 层之后，就有了雅称 HTTPS。通常，使用 SSL 指代 SSL/TLS 层。

注意：

- 建立 SSL 加密通信，认证并不是必须的。
- 但常见的情况是，SSL client 会认证 SSL server 的身份，这在本文中称为 “单向认证”。
- 同样，SSL server 也可以认证 SSL client 的证书，双方互相认证对方的证书，在本文中称为 “双向认证”。
- CA（Certificate Authority）是负责发放和管理数字证书的权威机构，并作为 SSL 证书认证中受信任的第三方。

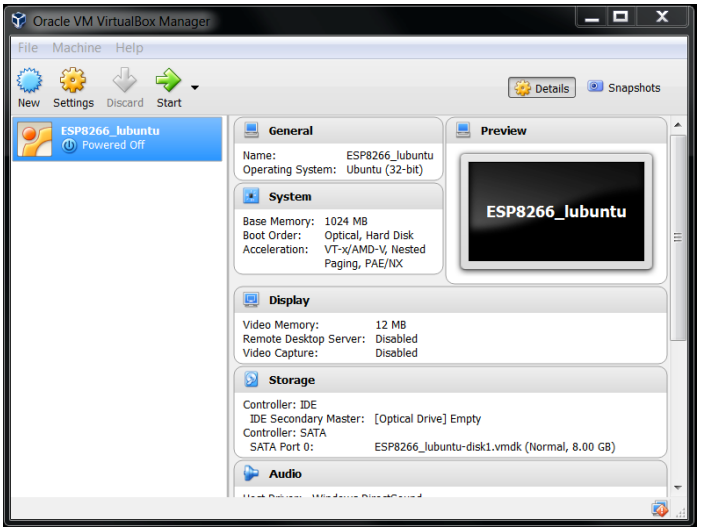
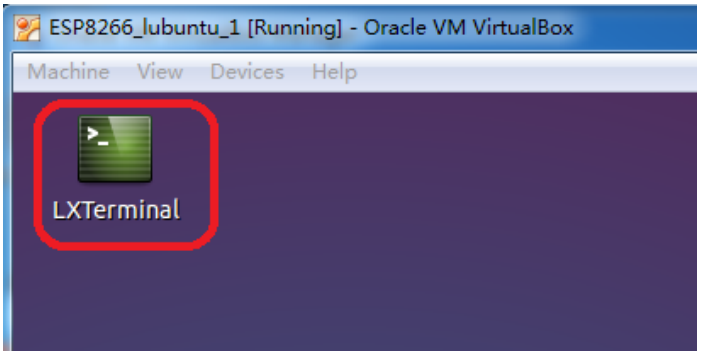
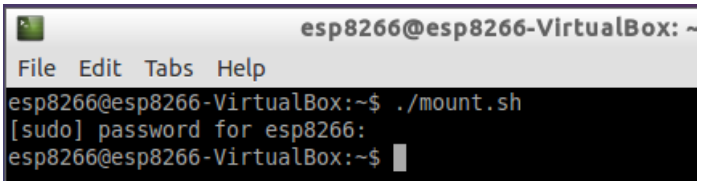
> 📖 **说明：**
>
> - 参考阅读资料 http://blog.csdn.net/ustccw/article/details/76691248。
> - 名词解释：
>   - 单向认证：仅 *SSL client* 认证 *SSL server* 的证书。
>   - 双向认证：*SSL client* 与 *SSL server* 互相认证对方的证书。

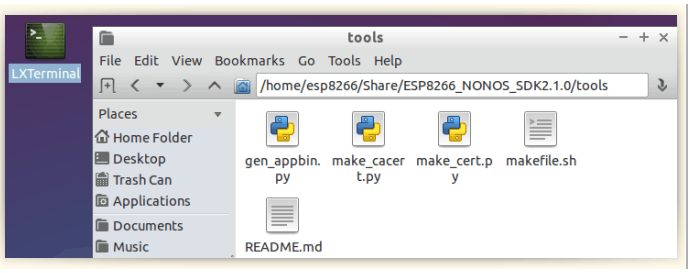
本文主要介绍基于 *ESP8266_NONOS_SDK* SSL 加密的证书认证使用方法，将分别介绍 ESP8266 作为 SSL server 和 ESP8266 作为 SSL client 的使用方法。

- ESP8266 作为 SSL server 时，
  - 单向认证：ESP8266 把自己的证书传给 SSL client，由 SSL client 选择是否校验 ESP8266 的证书。
  - 双向认证：在单向认证的基础上，ESP8266 还要求 SSL client 提供它的证书，由 ESP8266 认证 SSL client 是否可信。
- ESP8266 作为 SSL client 是更常用的情况，
  - 单向认证：ESP8266 将接收 SSL server 传来的服务器证书，并选择是否校验服务器的证书。
  - 双向认证：在单向认证的基础上，ESP8266 还需提供自己的证书给 SSL server，让 SSL server 选择是否校验 ESP8266。

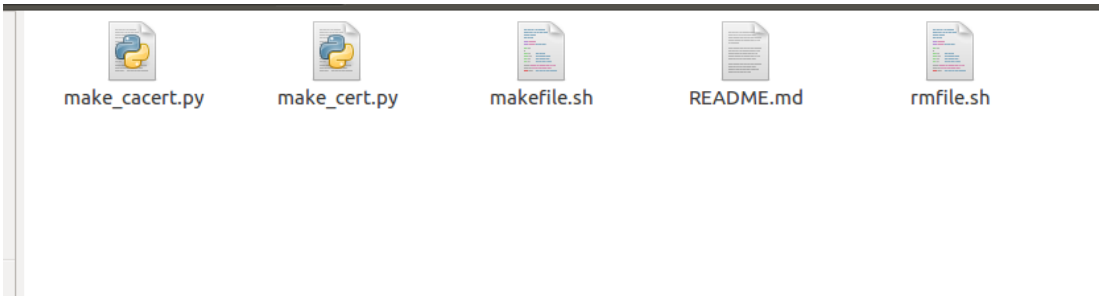
2. 编译环境与工具

下表介绍如何在 Windows 系统通过虚拟机搭建 Linux 编译环境。

步骤	结果
1. 进入Windows 系统安装 lubuntu 虚拟机。	
• 详细可参考 *ESP8266 SDK 入门指南* 搭建编译环境。	
2. 挂载共享路径。	
• 打开虚拟机桌面的 ***LXTerminal***。	
• 输入指令：***./mount.sh*** • 输入密码： ***espressif***	

步骤	结果
• 将 ***ESP8266_NONOS_SDK*** 拷贝到 lubuntu 虚拟机共享路径下； • 在虚拟机打开 ESP8266_NONOS_SDK/tools，看到 ***makefile.sh*** 代表挂载成功。	

在 *ESP8266_NONOS_SDK/tools* 中提供了 SSL 证书生成工具：



- ***makefile.sh***：SSL 证书格式转化和生成脚本。
  - ***make_cacert.py*** 和 ***make_cert.py*** 为 SSL 证书格式转化和生成使用的相关工具。
- ***rmfile.sh***：删除产生过的所有文件。

3. ESP8266 作为 SSL Server

ESP8266 作为 SSL server 的应用说明如下：

- 必须生成并包括 SSL 加密所需的头文件 ***cert.h*** 和 ***private_key.h***。
- CA 认证默认关闭，用户可调用接口 ***espconn_secure_ca_enable*** 使能，并将 CA 证书转化为 ***esp_ca_cert.bin*** 文件烧录.

开发者可参考 ***ESP8266_NONOS_SDK/examples/***IoT_Demo 中 `#define SERVER_SSL_ENABLE` 宏定义的代码，实现 SSL server 功能。

3.1. 证书制作

根据实际情况选择以下其中一种方式，生成作为 SSL server 时基本加密所需的头文件 ***cert.h*** 和 ***private_key.h***，以及 ***esp_ca_cert.bin***（仅 CA 认证时需要烧录）。

3.1.1. 无正式 CA 机构颁发的证书

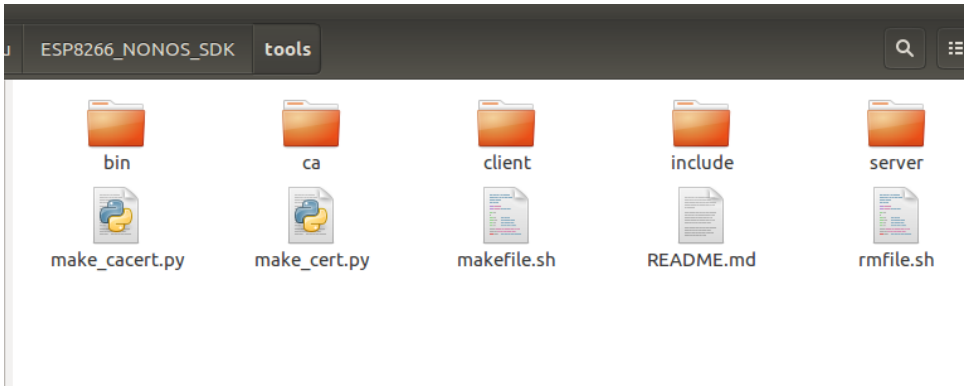
如果您没有正式 CA 机构颁发的证书，***ESP8266_NONOS_SDK/tools*** 中提供了 ***makefile.sh*** 工具，可以生成自签证书（即自己作为 CA，仅供测试使用），使用自签根证书给自己颁发服务器证书。

生成步骤如下：

1. 修改 ***makefile.sh*** 中的 ***CN*** 字段，具体将 ***192.168.111.100*** 改为 ESP8266 的实际 IP 地址。
2. 执行命令 ***makefile.sh*** 即可自动生成所有相关加密文件。

```
./makefile.sh
```

生成结果如下图：

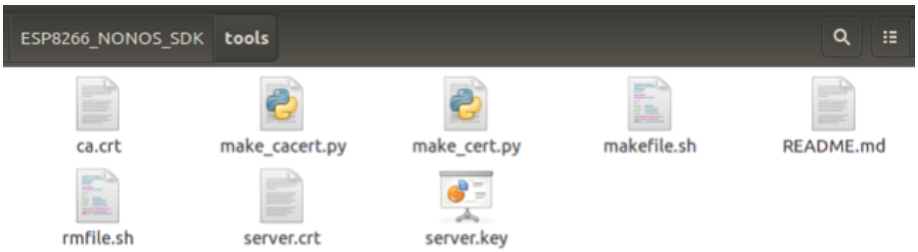


- 生成 SSL 加密所需的头文件 ***cert.h*** 和 ***private_key.h*** 位于 ***include*** 目录中。
- 生成 CA 认证所需的 ***esp_ca_cert.bin*** 位于 ***bin*** 目录下。

> 📖 **说明：**
>
> - ***ca*** 目录中为自签的证书。
> - 用户可根据加密需要，将 ***makefile.sh*** 中默认的 *1024* 位加密改为 *512* 位或其他。

3.1.2. 有私钥和正式 CA 机构颁发的证书

如果您有私钥 ***server.key***、CA 机构的正式证书 ***ca.crt***，及其颁发的 ***server.crt***，请将这三个文件拷贝至 ***ESP8266_NONOS_SDK/tools*** 目录下。如下图所示：

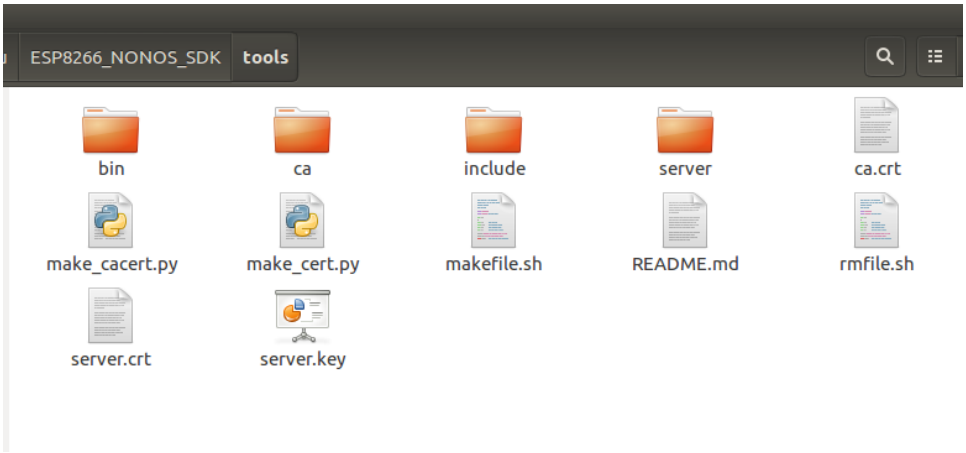


> ⚠ **注意：**
>
> - 证书名称如果与示例不符，则必须对应重命名为 *server.key, ca.crt, server.crt*。
> - 请确保 *ca.crt* 和 *server.crt* 为 *PEM* 格式。

直接执行命令 makefile.sh 即可自动生成所有相关加密文件。

```
./makefile.sh
```

生成结果如下图：



- 生成 SSL 加密所需的头文件 cert.h 和 private_key.h 位于 include 目录中。

- 生成 CA 认证所需的 esp_ca_cert.bin 位于 bin 目录下。

3.2. 证书使用说明

开发者请参考 ***IOT_Demo*** 中 `#define SERVER_SSL_ENABLE` 宏定义的代码来实现 SSL 相关功能。

注意事项如下：

- 开发者必须调用 ***espconn_secure_set_default_certificate*** 传入证书 ***cert.h***。
- 开发者必须调用 ***espconn_secure_set_default_private_key*** 传入密钥 ***private_key.h***。
- 如果使用 CA 认证，则还需烧录 CA 证书，具体如下：
  - 调用 ***espconn_secure_ca_enable*** 并指定证书位置，详细见**第 5 章软件接口**。
  - 烧录 CA 证书 ***esp_ca_cert.bin*** 到 ***espconn_secure_ca_enable*** 指定的位置。
- SSL 功能需要占用大量内存，请开发者在上层应用程序确保内存足够。
  - 在将 SSL 缓存设置为 8 KB (***espconn_secure_set_size***) 的情况下，SSL 功能至少需要 22 KB 的空间。
  - 由于服务器的证书大小不同，所需空间可能更大。
  - 如果内存不足，将导致 SSL 握手失败。
- 如果使能 SSL 双向认证功能，***espconn_secure_set_size*** 最大仅支持设置为 3,072 字节，在内存不足的情况下，SSL 缓存的空间必须设置到更小。

4. ESP8266 作为 SSL Client

ESP8266 作为 SSL client 时，用户可按实际使用情景，生成 SSL 加密所需的证书文件。

- 单向认证（仅作为 client 的 ESP8266 校验服务器合法性）：
  - 调用接口 ***espconn_secure_ca_enable*** 使能 CA 认证（默认状态下，CA 认证禁用）。
  - 生成 CA 证书文件 ***esp_ca_cert.bin***，并将其烧录至 ***espconn_secure_ca_enable*** 指定位置。
- 双向认证（ESP8266 与服务器互相校验对方证书的合法性）：
  - 需生成 CA 证书文件 ***esp_ca_cert.bin*** 以及 SSL client 证书私钥文件 ***esp_cert_private_key.bin***。

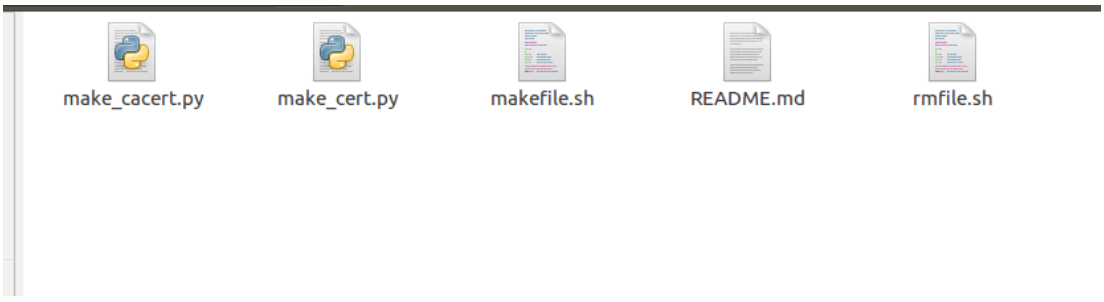
用户可以参考 *esp_mqtt_demo* 以及其中 `#define MQTT_SSL_ENABLE` 宏定义的代码，实现 SSL client 功能。

4.1. 证书制作

请根据实际情况选择以下其中一种方式，生成 SSL 加密所需的证书。

4.1.1. 无正式 CA 机构的证书

如果您没有任何正式的 CA 机构颁发的证书，***ESP8266_NONOS_SDK/tools*** 中提供了 makefile.sh 工具，用于生成自签 CA（***ca.crt*** + ***ca.key***），并使用自签 CA 给自己颁发证书，用于测试。

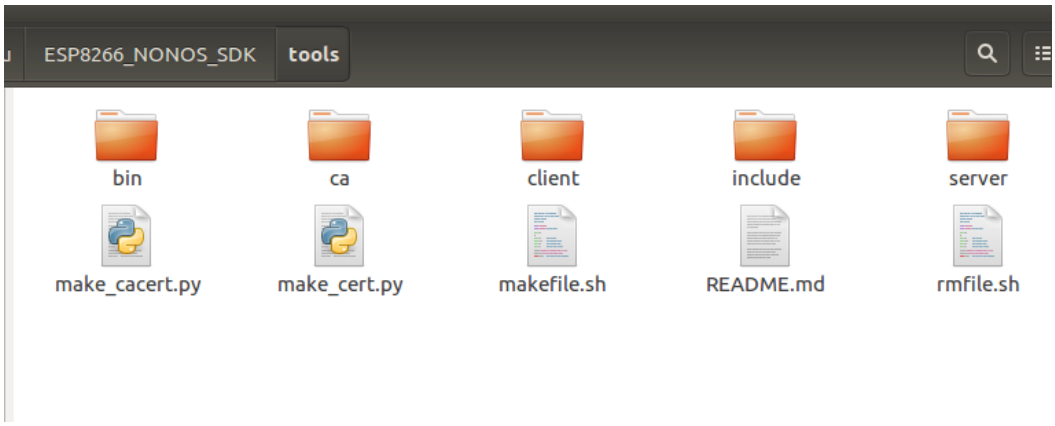


生成步骤如下：

1. 修改 ***makefile.sh*** 中 的 ***CN*** 字段，由 ***192.168.111.100*** 改为实际主机的 IP 地址。
2. 执行命令 ***makefile.sh*** 即可自动生成所有相关加密文件。

```
./makefile.sh
```

生成结果如下图：

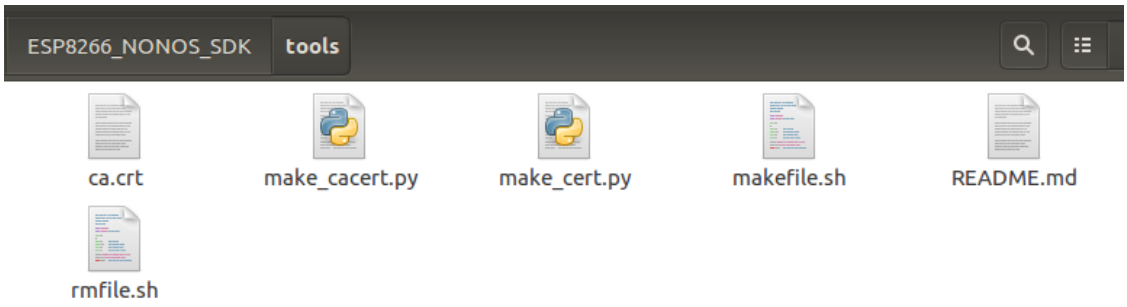


生成的 SSL 加密所需的 CA 证书文件 ***esp_ca_cert.bin*** 以及 client 证书私钥文件 ***esp_cert_private_key.bin*** 位于 ***bin*** 目录下。

> 📖 **说明：**
>
> - ***ca*** 目录中为自签的证书。
> - 用户可根据加密需要，将 ***makefile.sh*** 中默认的 *1024* 位加密改为 *512* 位或其他。

4.1.2. 仅有正式 CA 机构的证书 ca.crt

如果您仅有正式的 CA 机构的证书 ***ca.crt***，那么将 ***ca.crt*** 拷贝至 ***ESP8266_NONOS_SDK/tools*** 目录下，如图：

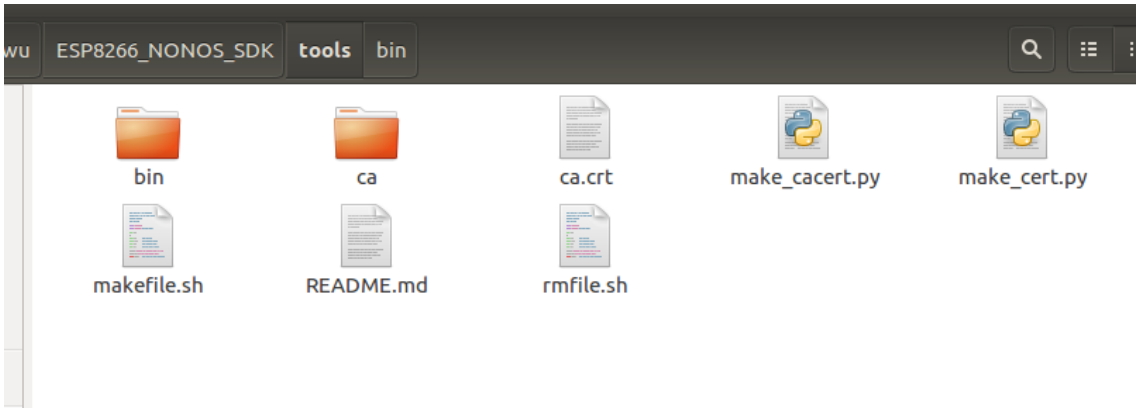


> ⚠ **注意：**
>
> - 证书名称如果与示例不符，则必须对应重命名为 ***ca.crt***。
> - 请确保 ***ca.crt*** 为 *PEM* 格式。

直接执行命令 ***makefile.sh*** 即可自动生成所有相关加密文件。

```
./makefile.sh
```

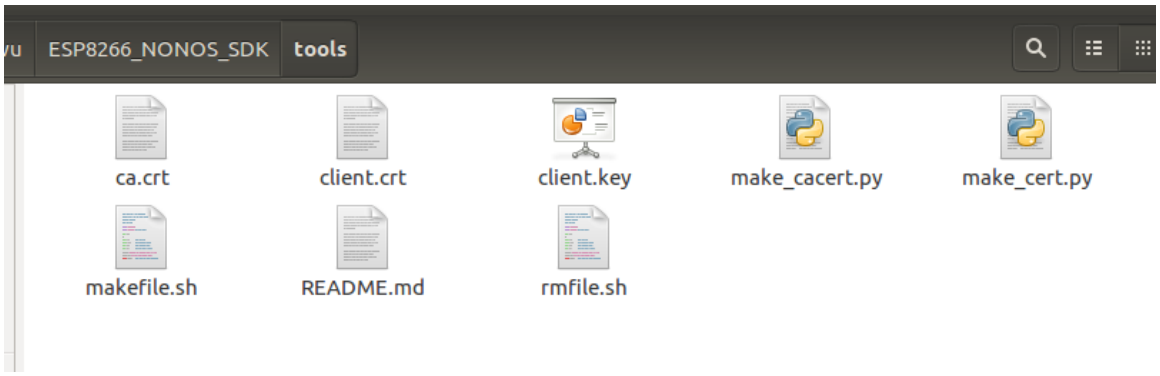
生成结果如下图：



生成单向认证所需的 CA 证书文件 ***esp_ca_cert.bin*** 位于 bin 目录下。

4.1.3. 有私钥和正式 CA 机构颁发的证书

如果您有私钥 ***client.key***，并且有正式的 CA 机构的证书 ***ca.crt*** 及其颁发的 ***client.crt***，那么将 ***client.key***，***ca.crt*** 和 ***client.crt*** 拷贝至 ***ESP8266_NONOS_SDK/tools*** 目录下，如图：



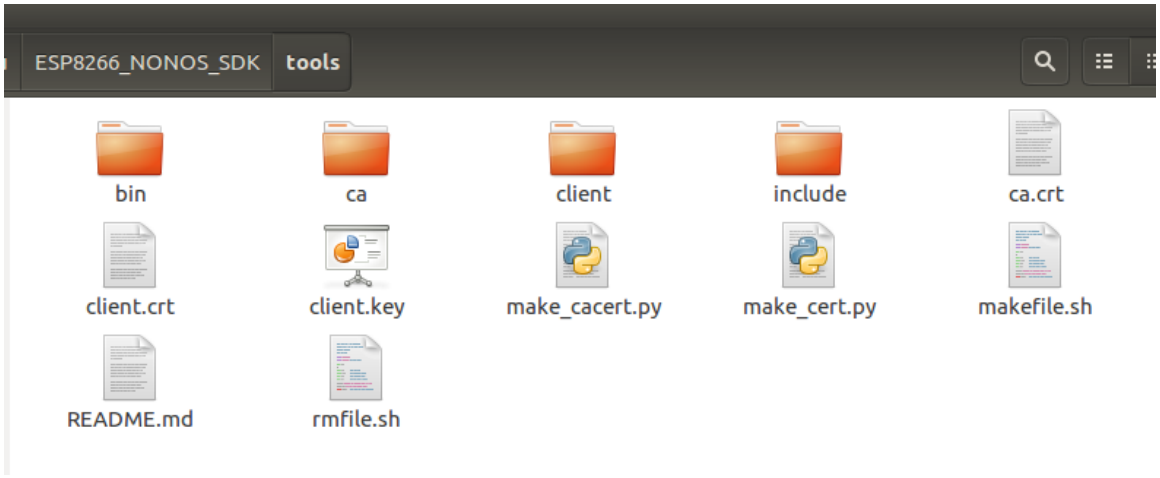
⚠ 注意：

- 证书名称如果与示例不符，则必须对应重命名为 ***client.key***, ***ca.crt***, ***client.crt***。
- 请确保 ***ca.crt*** 和 ***client.crt*** 为 *PEM* 格式。

直接执行命令 ***makefile.sh*** 即可自动生成所有相关加密文件。

```
./makefile.sh
```

生成结果如下图：



生成的 SSL 加密所需的 CA 证书文件 ***esp_ca_cert.bin*** 以及 client 证书私钥文件 ***esp_cert_private_key.bin*** 位于 ***bin*** 目录下。

4.2. 证书使用说明

开发者请参考 *esp_mqtt_proj* 以及其中 `#define MQTT_SSL_ENABLE` 宏定义的代码来实现 SSL 相关功能。

注意事项如下：

- 如需使能单向认证，即仅 ESP8266 校验服务器，则设置如下：
  - 必须调用接口 ***espconn_secure_ca_enable*** 使能 CA 认证。
  - 必须烧录 ***esp_ca_cert.bin***，烧录的位置由 ***espconn_secure_ca_enable*** 的第二个参数决定。
- 如需使能双向认证，即 ESP8266 与服务器互相校验对方的证书，则在上述单向认证的基础上，增加如下设置：
  - 除了接口 ***espconn_secure_ca_enable***，接口 ***espconn_secure_cert_req_enable*** 也应调用使能 CA 认证。
  - 必须烧录 ***esp_ca_cert.bin***，烧录的位置由 ***espconn_secure_ca_enable*** 的第二个参数决定。
  - 烧录 ***esp_cert_private_key.bin***，烧录位置由 ***espconn_secure_cert_req_enable*** 第二个参数决定。
- SSL 功能需要占用大量内存，请开发者在上层应用程序确保内存足够。
  - 在将 SSL 缓存设置为 8 KB (***espconn_secure_set_size***) 的情况下，SSL 功能至少需要 22 KB 的空间。

  - 由于服务器的证书大小不同，所需空间可能更大。
  - 如果内存不足，将导致 SSL handshake 失败。

- 如果使能 SSL 双向认证功能，***espconn_secure_set_size*** 最大仅支持设置为 3072 字节，在内存不足的情况下，SSL 缓存的空间必须设置到更小。如果内存不足，将导致 SSL handshake 失败。

5. 软件接口

SSL 软件接口与普通 TCP 软件接口，在 SDK 底层是两套不同的处理流程，因此，请不要混用两种软件接口。SSL 连接时，仅支持使用：

- ***espconn_secure_XXX*** 系列接口；
- ***espconn_regist_XXXcb*** 系列注册回调的接口，除了 ***espconn_regist_write_finish***；
- ***espconn_port*** 获得一个空闲端口。

本文仅介绍 ***espconn_secure_XXX*** 系列接口，更多的软件接口介绍，请参考《*ESP8266 Non-OS SDK API 参考*》。

5.1. *espconn_secure_accept*

功能	创建 SSL TCP server，侦听 SSL 握手
函数定义	***sint8 espconn_secure_accept(struct espconn *espconn)***
参数	***struct espconn *espconn***：对应网络连接的结构体

返回	• ***0***：成功
	• 其它：失败
	  - ***ESPCONN_MEM***：空间不足
	  - ***ESPCONN_ISCONN***：连接已经建立
	  - ***ESPCONN_ARG***：非法参数，未找到参数 ***espconn*** 对应的 TCP 连接

注意	• 目前仅支持建立一个 SSL server，本接口只能调用一次，并且仅支持连入一个 SSL client。 • 如果 SSL 加密一包数据大于 ***espconn_secure_set_size*** 设置的缓存空间（默认为 2 KB），ESP8266 无法处理，SSL 连接断开，调用 ***espconn_reconnect_callback***。 • SSL 相关接口与普通 TCP 接口底层处理不一致，请不要混用。SSL 连接时，仅支持使用 ***espconn_secure_XXX*** 系列接口、***espconn_regist_XXXcb*** 系列接口（注册回调函数），以及 ***espconn_port*** 接口（获得一个空闲端口）。 • 如需创建 SSL server，必须先调用 ***espconn_secure_set_default_certificate*** 和 ***espconn_secure_set_default_private_key*** 传入证书和密钥。

5.2. *espconn_secure_delete*

功能	删除 ESP8266 作为 SSL server 的连接。
函数定义	***sint8 espconn_secure_delete(struct espconn *espconn)***
参数	***struct espconn *espconn***：对应网络连接的结构体

返回	• ***0***：成功
	• 其它：失败
	  - ***ESPCONN_ARG***：非法参数，未找到参数 ***espconn*** 对应的 TCP 连接
	  - ***ESPCONN_INPROGRESS***：参数 ***espconn*** 对应的 SSL 连接仍未断开，请先调用 ***espconn_secure_disconnect*** 断开连接，再进行删除。

5.3. *espconn_secure_set_size*

功能	设置加密 (SSL) 数据缓存空间的大小
函数定义	***bool espconn_secure_set_size (uint8 level, uint16 size)***

参数	• ***uint8 level***：设置 ESP8266 SSL server/client
	  - ***0x01***：SSL client
	  - ***0x02***：SSL server
	  - ***0x03***：SSL client 和 SSL server
	• ***uint16 size***：加密数据缓存的空间大小，取值范围：1 ~ 8,192 字节，默认值为 2,048 字节

返回	• ***true***：成功 • ***false***：失败
注意	默认缓存大小为 2KB； 如需更改，请在加密 (SSL) 连接建立前调用： • 在 ***espconn_secure_accept***（ESP8266 作为 TCP SSL server）之前调用； • 或者 ***espconn_secure_connect***（ESP8266 作为 TCP SSL client）之前调用。

5.4. *espconn_secure_get_size*

功能	查询加密 (SSL) 数据缓存空间的大小
函数定义	***sint16 espconn_secure_get_size (uint8 level)***
参数	***uint8 level***：设置该缓存空间在 ESP8266 SSL 为 server、client 或同时有效 • ***0x01***：SSL client • ***0x02***：SSL server • ***0x03***：SSL client 和 SSL server
返回	加密 (SSL) 数据缓存空间的大小

5.5. *espconn_secure_connect*

功能	加密 (SSL) 连接到 TCP SSL server（ESP8266 作为 TCP SSL client）
函数定义	***sint8 espconn_secure_connect (struct espconn *espconn)***
参数	***struct espconn *espconn***：对应网络连接的结构体

返回	• ***0***：成功
	• 其它：失败
	  - ***ESPCONN_MEM***：空间不足
	  - ***ESPCONN_ISCONN***：连接已经建立
	  - ***ESPCONN_ARG***：非法参数，未找到参数 ***espconn*** 对应的 TCP 连接

注意	• 如果 ***espconn_secure_connect*** 失败，返回非零值，连接未建立，不会进行任何 ***espconn*** callback。 • 目前 ESP8266 作为 SSL client 仅支持一个连接，本接口只能调用一次，或者调用 ***espconn_secure_disconnect*** 断开前一次连接，才可以再次调用本接口建立 SSL 连接。 • 如果 SSL 加密一包数据大于 ***espconn_secure_set_size*** 设置的缓存空间，ESP8266 无法处理，SSL 连接断开，进入 ***espconn_reconnect_callback***。 • SSL 相关接口与普通 TCP 接口底层处理不一致，请不要混用。SSL 连接时，仅支持使用 ***espconn_secure_XXX*** 系列接口、***espconn_regist_XXXcb*** 系列接口（注册回调函数），以及 ***espconn_port*** 接口（获得一个空闲端口）。

5.6. *espconn_secure_send*

功能	发送加密数据 (SSL)
函数定义	***sint8 espconn_secure_send (*** ***struct espconn *espconn,*** ***uint8 *psent,*** ***uint16 length*** ***)***
参数	***struct espconn *espconn***：对应网络连接的结构体 ***uint8 *psent***：发送的数据 ***uint16 length***：发送的数据长度
返回	• ***0***：成功 • ***ESPCONN_ARG***：未找到参数 ***espconn*** 对应的 TCP 连接
注意	• 请在上一包数据发送完成，进入 ***espconn_sent_callback*** 后，再发下一包数据。 • 每一包数据明文的上限值为 1,024 字节，加密后的报文上限值是 1,460 字节。

5.7. *espconn_secure_disconnect*

功能	断开加密 TCP 连接 (SSL)
函数定义	***sint8 espconn_secure_disconnect(struct espconn *espconn)***
参数	***struct espconn *espconn***：对应网络连接的结构体
返回	• ***0***：成功 • ***ESPCONN_ARG***：未找到参数 ***espconn*** 对应的 TCP 连接
注意	请勿在 ***espconn*** 的任何 ***callback*** 中调用本接口断开连接。如有需要，请使用 ***system_os_task*** 和 ***system_os_post***，触发任务 ***espconn_secure_disconnect***，断开连接。

5.8. *espconn_secure_ca_enable*

功能	开启 SSL CA 认证功能
函数定义	***bool espconn_secure_ca_enable (uint8 level, uint32 flash_sector)***

参数	• ***uint8 level***：设置 ESP8266 SSL server/client
	  - ***0x01***：SSL client
	  - ***0x02***：SSL server
	  - ***0x03***：SSL client 和 SSL server
	• ***uint32 flash_sector***：设置 CA 证书 ***esp_ca_cert.bin*** 烧录到 flash 的位置。例如，参数传入 ***0x7B***，则对应烧录到 flash 的 ***0x7B000***。

返回	• ***true***：成功 • ***false***：失败

注意	• CA 认证功能，默认关闭。
	• 如需调用本接口，请在加密 (SSL) 连接建立前调用：
	  - 在 ***espconn_secure_accept***（ESP8266 作为 TCP SSL server）之前调用；
	  - 或者 ***espconn_secure_connect***（ESP8266 作为 TCP SSL client）之前调用

5.9. *espconn_secure_ca_disable*

功能	关闭 SSL CA 认证功能
函数定义	***bool espconn_secure_ca_disable (uint8 level)***
参数	• ***uint8 level***：设置该缓存空间在 ESP8266 SSL 为 server、client 或同时有效 - ***0x01***：SSL client - ***0x02***：SSL server - ***0x03***：SSL client 和 SSL server
返回	• ***true***：成功 • ***false***：失败
注意	• CA 认证功能，默认关闭。 • 如需调用本接口，请在加密 (SSL) 连接建立前调用： - 在 ***espconn_secure_accept***（ESP8266 作为 TCP SSL server）之前调用； - 在 ***espconn_secure_connect***（ESP8266 作为 TCP SSL client）之前调用。

5.10. *espconn_secure_cert_req_enable*

功能	使能 ESP8266 作为 SSL client 时的证书认证功能
函数定义	***bool espconn_secure_cert_req_enable (uint8 level, uint32 flash_sector)***
参数	• ***uint8 level***：ESP8266 作为 SSL client，仅支持设置为 ***0x01*** • ***uint32 flash_sector***：设置密钥 ***esp_cert_private_key.bin*** 烧录到 Flash 的位置，例如，参数传入 0x7A，则对应烧录到 Flash 0x7A000。请注意，不要覆盖到代码或系统参数区域。
返回	• ***true***：成功 • ***false***：失败
注意	• 证书认证功能，默认关闭。如果服务器端不要求认证证书，则无需调用本接口。 • 如需调用本接口，请在 ***espconn_secure_connect*** 之前调用。

5.11. *espconn_secure_cert_req_disable*

功能	关闭 ESP8266 作为 SSL client 时的证书认证功能
函数定义	***bool espconn_secure_ca_disable (uint8 level)***
参数	***uint8 level***：仅支持设置为 ***0x01*** ESP8266 作为 SSL client
返回	• ***true***：成功 • ***false***：失败
注意	证书认证功能，默认关闭。

5.12. *espconn_secure_set_default_certificate*

功能	设置 ESP8266 作为 SSL server 时的证书
函数定义	***bool espconn_secure_set_default_certificate (const uint8_t* certificate, uint16_t length)***
参数	***const uint8_t* certificate***：证书指针 ***uint16_t length***：证书长度
返回	• ***true***：成功 • ***false***：失败
注意	• ***ESP8266_NONOS_SDK/examples/IoT_Demo***中提供使用示例 • 本接口必须在 ***espconn_secure_accept*** 之前调用，传入证书信息

5.13. *espconn_secure_set_default_private_key*

功能	设置 ESP8266 作为 SSL server 时的密钥
函数定义	***bool espconn_secure_set_default_private_key (const uint8_t* key, uint16_t length)***
参数	• ***const uint8_t* key***：密钥指针 • ***uint16_t length***：密钥长度
返回	• ***true***：成功 • ***false***：失败
注意	• ***ESP8266_NONOS_SDK/examples/IoT_Demo*** 中提供使用示例 • 本接口必须在 ***espconn_secure_accept*** 之前调用，传入密钥信息

乐鑫 IoT 团队

www.espressif.com