

# ESP8266 SSL 加密 使用手册



版本 1.5

版权 © 2017

# 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
2017.08	V1.1	文档更新。

## 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.....	1
2. 环境搭建 .....	3
3. ESP8266 作为 SSL Server .....	5
3.1. 证书制作 .....	5
3.1.1. 无正式 CA 机构颁发的证书 .....	5
3.1.2. 有私钥和正式 CA 机构颁发的证书 .....	6
3.2. 证书使用说明 .....	7
4. ESP8266 作为 SSL Client .....	8
4.1. 证书制作 .....	8
4.1.1. 无正式 CA 机构的证书 .....	8
4.1.2. 仅有正式 CA 机构的证书 ca.crt .....	9
4.1.3. 有私钥和正式 CA 机构颁发的证书 .....	10
4.2. 证书使用说明 .....	11
5. 软件接口 .....	13
5.1. espconn_secure_accept .....	13
5.2. espconn_secure_delete .....	13
5.3. espconn_secure_set_size .....	14
5.4. espconn_secure_get_size .....	14
5.5. espconn_secure_connect .....	15
5.6. espconn_secure_send .....	15
5.7. espconn_secure_disconnect .....	16
5.8. espconn_secure_ca_enable.....	16
5.9. espconn_secure_ca_disable .....	16
5.10. espconn_secure_cert_req_enable .....	17
5.11. espconn_secure_cert_req_disable.....	17
5.12. espconn_secure_set_default_certificate.....	17
5.13. espconn_secure_set_default_private_key .....	18



# 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)。它们的作用，就是在明文的 TCP 以上层和 TCP 层之间加一层加密层，用以保证上层信息传输的安全。例如，HTTP 协议是明文传输，加上 SSL 层之后，就有了雅称 HTTPS。它的发展依次经历了下面几个时期：

- SSL1.0：已废除
- SSL2.0：RFC6176，已废除
- SSL3.0：RFC6101，基本废除
- TLS1.0：RFC2246，目前大都支持此种方式
- TLS1.1：RFC4346
- TLS1.2：RFC5246，没有广泛使用
- TLS1.3：IETF 正在酝酿中

通常，使用 SSL 指代 SSL/TLS 层。

## 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 的证书是否合法。
  - 双向认证 (CA 认证)：SSL client 与 SSL server 均需认证对方的证书是否合法。

本文主要介绍基于 [ESP8266\\_NONOS\\_SDK](#) 的 SSL 加密使用方法，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 证书的合法性。
  - 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 决定是否校验 SSL client 的合法性。
- 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，由 ESP8266 选择是否校验服务器证书的合法性。



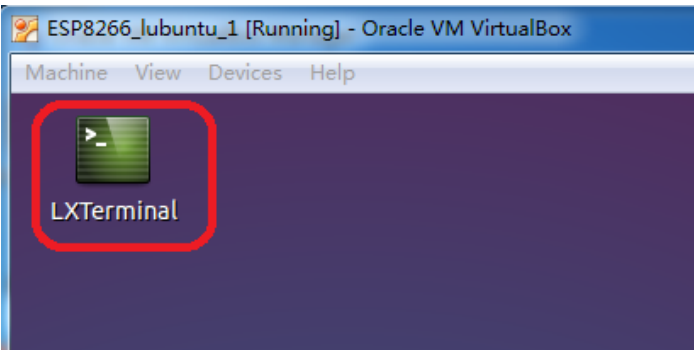
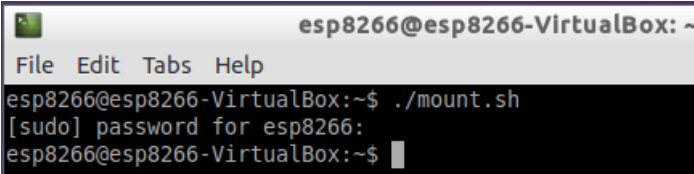
- 双向认证：在单向认证的基础上，ESP8266 还需提供自己的证书给 SSL server，让 SSL server 选择是否校验 ESP8266 的合法性。



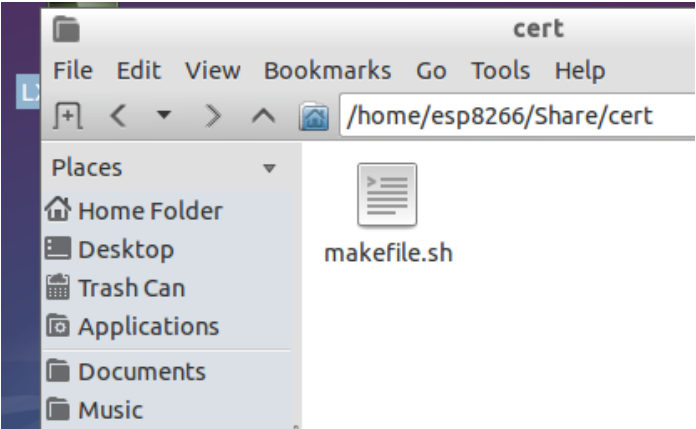
## 2.

# 环境搭建

如果用户使用的 Linux 系统，则可以跳过本章；如果用户使用的是 windows 系统，可参考本章搭建 Linux 编译环境。

步骤	结果
1. 进入Windows 系统安装 lubuntu 虚拟机并打开。	
2. 挂载共享路径。	
	



步骤	结果
<ul style="list-style-type: none"><li>将证书制作脚本“makefile.sh”拷贝到 lubuntu 虚拟机共享路径下。</li><li>在虚拟机打开共享文件夹，看到证书制作脚本，挂载成功。</li></ul>	



## 3. ESP8266 作为 SSL Server

ESP8266 作为 SSL server 的应用说明如下：

- 必须生成 SSL 加密所需的头文件 cert.h 和 private\_key.h。
- 双向认证功能（即 CA 认证）默认关闭，用户可调用接口 espconn\_secure\_ca\_enable 使能 CA 认证，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](#) 以及 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

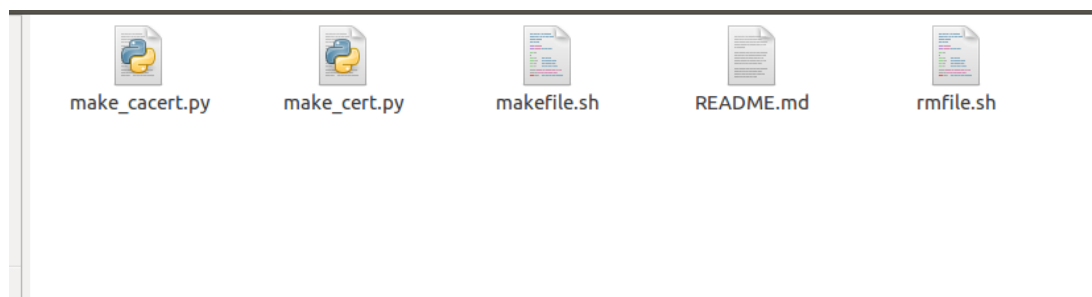
拷贝 make\_cacert.py，make\_cert.py 和 rmfile.sh 到 makefile.sh 同目录下。

- rmfile.sh 可删除产生过的所有文件。
- make\_cacert.py 和 make\_cert.py 为证书格式转化和生成用到的工具。

根据实际情况选择下列其中一种方式，生成作为 SSL server 时 SSL 加密所需的头文件 cert.h 和 private\_key.h，以及 esp\_ca\_cert.bin（仅双向认证时需要烧录）。

#### 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，我们在 ESP8266\_NONOS\_SDK/tools 中提供了如下图的工具，用于生成自签证书（即自己作为 CA，仅供测试使用），并使用自签 CA 给自己颁发服务器证书。



生成步骤如下：

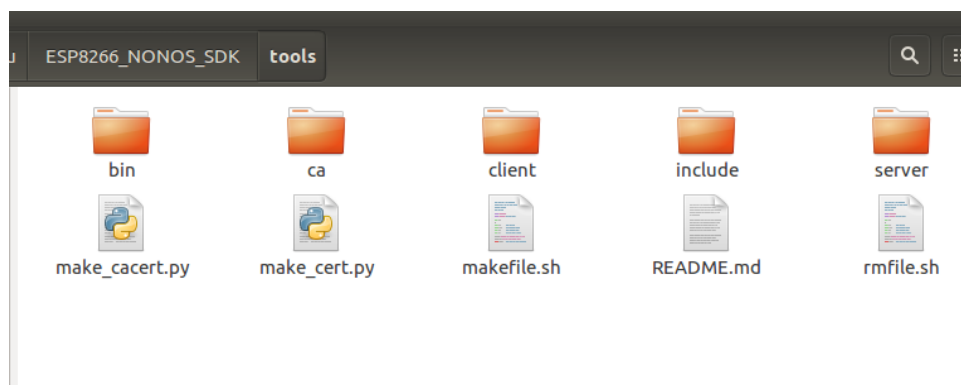
1. 修改 makefile.sh 中的 CN 字段，将 192.168.111.100 改为 ESP8266 实际 IP 地址。
2. 执行命令 makefile.sh 即可自动生成所有相关加密文件。

```
./makefile.sh
```





生成结果如下图：



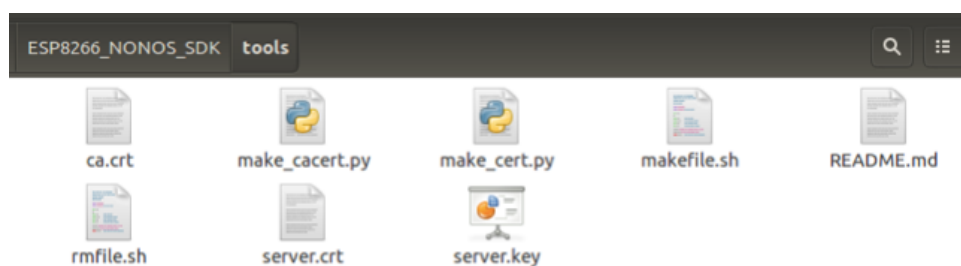
- 生成 SSL 加密所需的头文件 cert.h 和 private\_key.h 位于 include 目录中。
- 生成双向认证所需的 esp\_ca\_cert.bin 位于 bin 目录下。

其他说明：

- ca 目录为自签的 CA。
- 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，并且有正式的 CA 机构的证书 ca.crt 及其颁发的 server.crt，请将 server.key，ca.crt 和 server.crt 拷贝至 ESP8266\_NONOS\_SDK/tools 目录下。如下图所示：



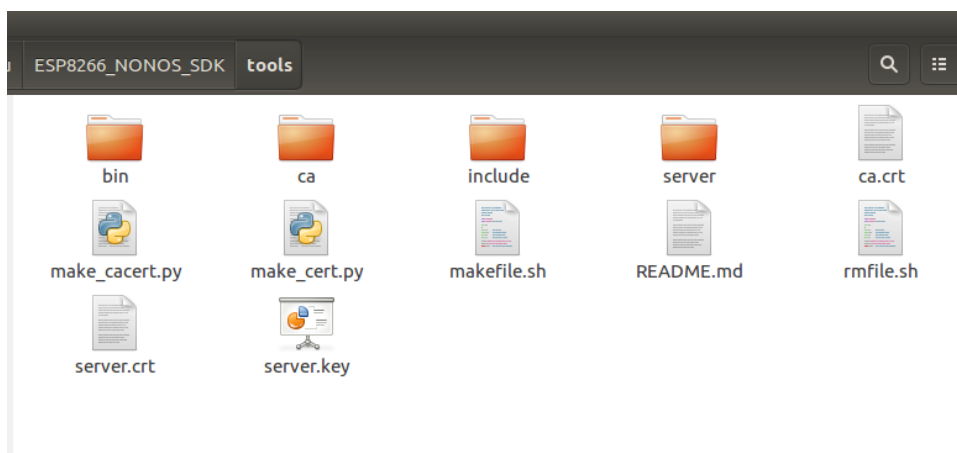
#### ⚠ 注意：

- 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。
- 生成双向认证所需的 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。
- 如果使用双向认证，则还需烧录 CA 证书，具体如下：
  - 调用 `espconn_secure_ca_enable` 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 8KB (`espconn_secure_set_size`) 的情况下，SSL 功能至少需要 22KB 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。



## 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 ESP8266 校验服务器合法性）：
  - 调用接口 `espconn_secure_ca_enable` 使能单向认证。
  - 生成并烧录 CA 证书文件 `esp_ca_cert.bin`。
- 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

拷贝 `make_cacert.py`，`make_cert.py` 和 `rmfile.sh` 到 `makefile.sh` 同目录下。

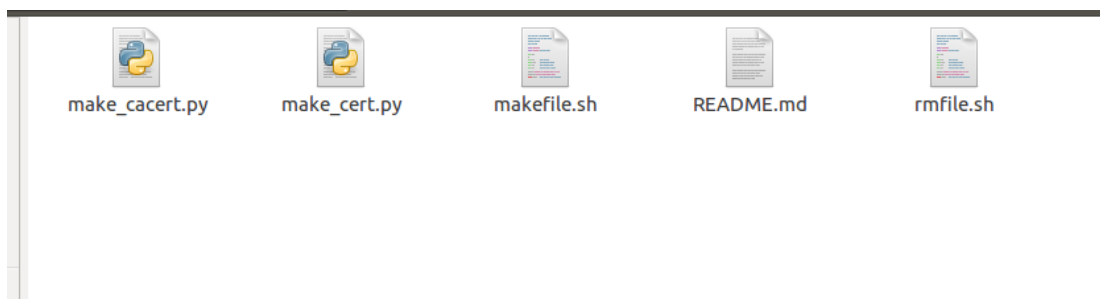
- `rmfile.sh` 可删除产生过的所有文件。
- `make_cacert.py` 和 `make_cert.py` 为证书格式转化和生成用到的工具。

根据实际情况选择下列其中一种方式，生成 SSL 加密所需的 bin 文件：

- 单向认证需生成并烧录 `esp_ca_cert.bin`。
- 双向认证需生成并烧录 `esp_ca_cert.bin` 和 `esp_cert_private_key.bin`。

#### 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，我们在 ESP8266\_NONOS\_SDK/tools 中提供了如下图的工具，用于生成自签 CA（`ca.crt` + `ca.key`），并使用自签 CA 给自己颁发证书。



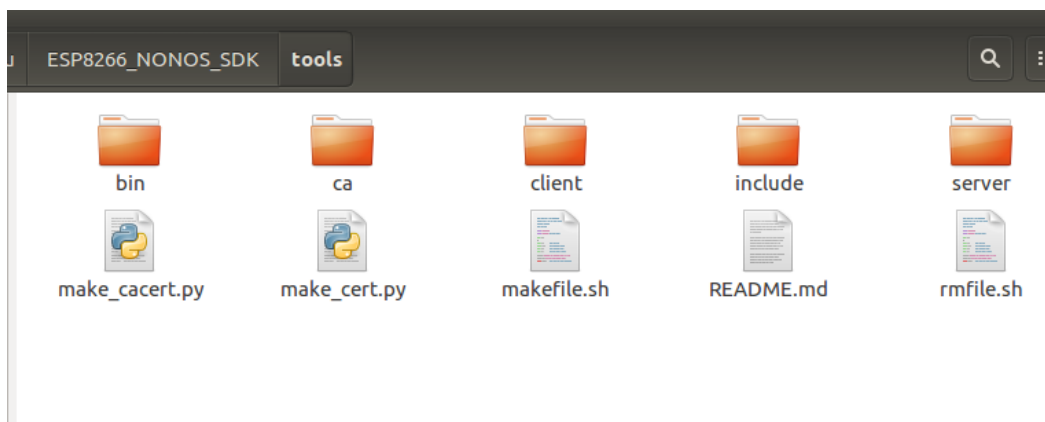


生成步骤如下：

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：



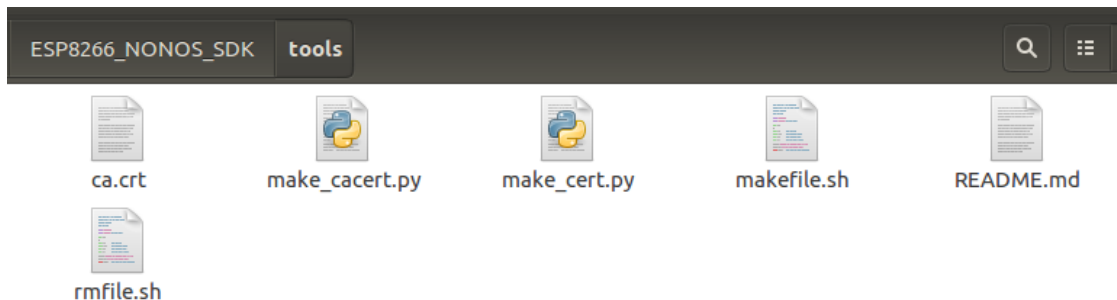
生成的 SSL 加密所需的 CA 证书文件 esp\_ca\_cert.bin 以及 client 证书私钥文件 esp\_cert\_private\_key.bin 位于 bin 目录下。

其他说明：

- ca 目录为自签的 CA。
- 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：



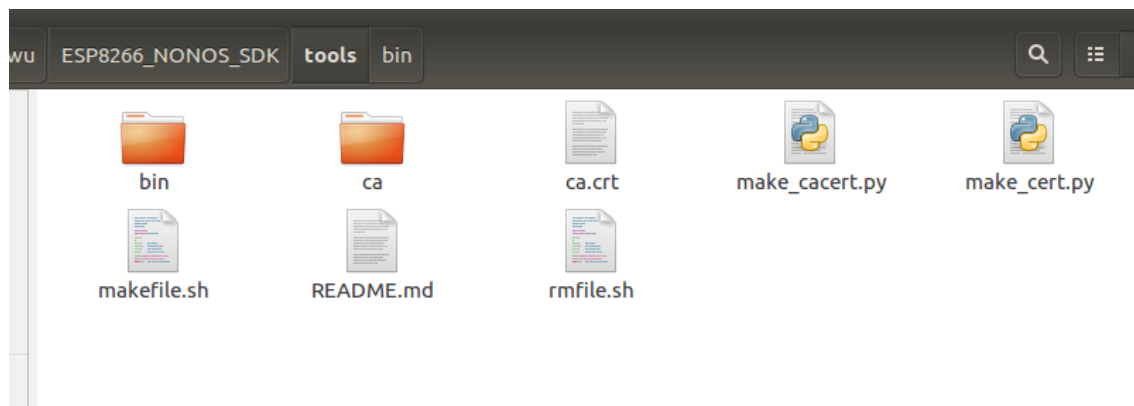
**⚠ 注意：**

- 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
```

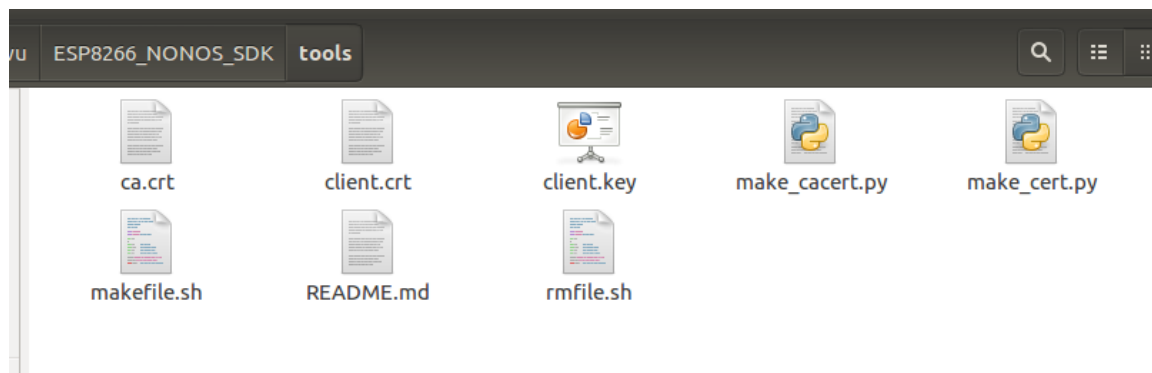
生成结果如下图：



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：**

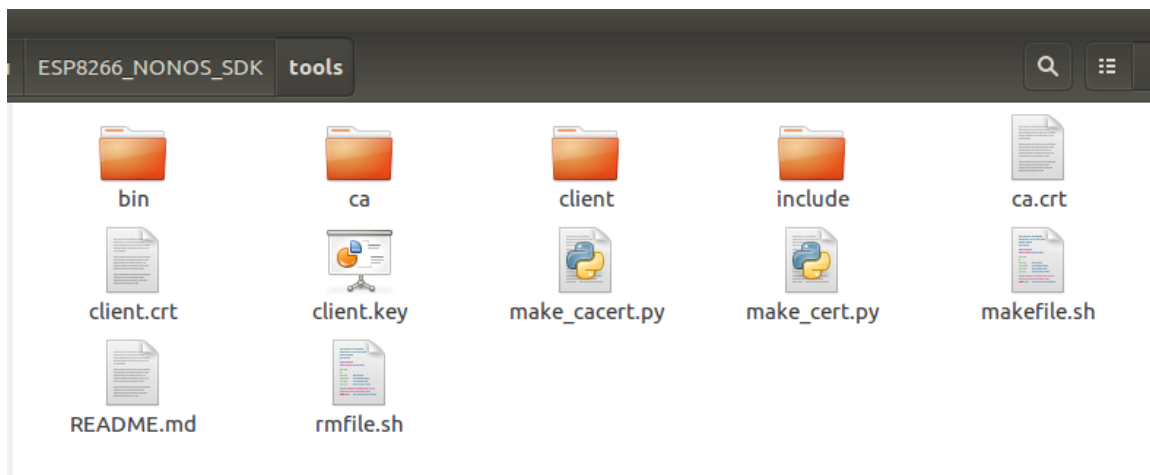
- 证书名称如果与示例不符，则必须对应重命名为 *client.key*，*ca.crt*，*client.crt*。
- 请确保 *ca.crt* 和 *server.crt* 为 PEM 格式。

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 ESP8266 校验服务器证书，则设置如下：
  - 必须调用接口 `espconn_secure_ca_enable` 使能认证。
  - 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：
  - 调用接口 `espconn_secure_cert_req_enable` 使能。
  - 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 8KB (`espconn_secure_set_size`) 的情况下，SSL 功能至少需要 22KB 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 API 说明文档。

## 5.1. espconn\_secure\_accept

功能	创建 SSL TCP server，侦听 SSL 握手
注意	<ul style="list-style-type: none"><li>• 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。</li><li>• 如果 SSL 加密一包数据大于 <code>espconn_secure_set_size</code> 设置的缓存空间，ESP8266 无法处理，SSL 连接断开，进入 <code>espconn_reconnect_callback</code>。</li><li>• 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 <code>espconn_secure_XXX</code> 系列接口和 <code>espconn_regist_XXXcb</code> 系列注册回调函数的接口，以及 <code>espconn_port</code> 获得一个空闲端口。</li><li>• 如需创建 SSL server，必须先调用 <code>espconn_secure_set_default_certificate</code> 和 <code>espconn_secure_set_default_private_key</code> 传入证书和密钥。</li></ul>
函数定义	<code>sint8 espconn_secure_accept(struct espconn *espconn)</code>
参数	<code>struct espconn *espconn</code> ：对应网络连接的结构体
返回	<code>0</code> ：成功 其它：失败，返回错误码 <ul style="list-style-type: none"><li>• <code>ESPCONN_ARG</code>：未找到参数 <code>espconn</code> 对应的 TCP 连接</li><li>• <code>ESPCONN_MEM</code>：空间不足</li><li>• <code>ESPCONN_ISCONN</code>：连接已经建立</li></ul>

## 5.2. espconn\_secure\_delete

功能	删除 ESP8266 作为 SSL server 的连接。
----	-------------------------------





函数定义	<code>sint8 espconn_secure_delete(struct espconn *espconn)</code>
参数	<code>struct espconn *espconn</code> : 对应网络连接的结构体
返回	<code>0</code> : 成功 其它: 失败, 返回错误码 <ul style="list-style-type: none"><li><code>ESPCONN_ARG</code>: 未找到参数 <code>espconn</code> 对应的 TCP 连接</li><li><code>ESPCONN_INPROGRESS</code>: 参数 <code>espconn</code> 对应的 SSL 连接仍未断开, 请先调用 <code>espconn_secure_disconnect</code> 断开连接, 再进行删除。</li></ul>

### 5.3. espconn\_secure\_set\_size

功能	设置加密 (SSL) 数据缓存空间的大小
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: <ul style="list-style-type: none"><li>在 <code>espconn_secure_accept</code> (ESP8266 作为 TCP SSL server) 之前调用;</li><li>或者 <code>espconn_secure_connect</code> (ESP8266 作为 TCP SSL client) 之前调用</li></ul>
函数定义	<code>bool espconn_secure_set_size (uint8 level, uint16 size)</code>
参数	<ul style="list-style-type: none"><li><code>uint8 level</code>: 设置 ESP8266 SSL server/client<ul style="list-style-type: none"><li><code>0x01</code>: SSL client</li><li><code>0x02</code>: SSL server</li><li><code>0x03</code>: SSL client 和 SSL server</li></ul></li><li><code>uint16 size</code>: 加密数据缓存的空间大小, 取值范围: 1 ~ 8192, 单位: 字节, 默认值为 2048</li></ul>
返回	<code>true</code> : 成功 <code>false</code> : 失败

### 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	<code>sint16 espconn_secure_get_size (uint8 level)</code>
参数	<ul style="list-style-type: none"><li><code>uint8 level</code>: 设置 ESP8266 SSL server/client<ul style="list-style-type: none"><li><code>0x01</code>: SSL client</li><li><code>0x02</code>: SSL server</li><li><code>0x03</code>: SSL client 和 SSL server</li></ul></li></ul>
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)
注意	<ul style="list-style-type: none"><li>如果 <code>espconn_secure_connect</code> 失败, 返回非零值, 连接未建立, 不会进入任何 <code>espconn</code> callback。</li><li>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 <code>espconn_secure_disconnect</code> 断开前一次连接, 才可以再次调用本接口建立 SSL 连接;</li><li>如果 SSL 加密一包数据大于 <code>espconn_secure_set_size</code> 设置的缓存空间, ESP8266 无法处理, SSL 连接断开, 进入 <code>espconn_reconnect_callback</code></li><li>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 <code>espconn_secure_XXX</code> 系列接口和 <code>espconn_regist_XXXcb</code> 系列注册回调函数的接口, 以及 <code>espconn_port</code> 获得一个空闲端口。</li></ul>
函数定义	<code>sint8 espconn_secure_connect (struct espconn *espconn)</code>
参数	<code>struct espconn *espconn</code> : 对应网络连接的结构体
返回	<code>0</code> : 成功 其它: 失败, 返回错误码 <ul style="list-style-type: none"><li><code>ESPCONN_ARG</code>: 未找到参数 <code>espconn</code> 对应的 TCP 连接</li><li><code>ESPCONN_MEM</code>: 空间不足</li><li><code>ESPCONN_ISCONN</code>: 连接已经建立</li></ul>

## 5.6. espconn\_secure\_send

功能	发送加密数据 (SSL)
注意	<ul style="list-style-type: none"><li>请在上一包数据发送完成, 进入 <code>espconn_sent_callback</code> 后, 再发下一包数据。</li><li>每一包数据明文的上限值为 1024 字节, 加密后的报文上限值是 1460 字节。</li></ul>
函数定义	<pre>sint8 espconn_secure_send (     struct espconn *espconn,     uint8 *psent,     uint16 length )</pre>
参数	<code>struct espconn *espconn</code> : 对应网络连接的结构体 <code>uint8 *psent</code> : 发送的数据 <code>uint16 length</code> : 发送的数据长度
返回	<code>0</code> : 成功 其它: 失败, 返回错误码 <code>ESPCONN_ARG</code> : 未找到参数 <code>espconn</code> 对应的 TCP 连接



## 5.7. espconn\_secure\_disconnect

功能	断开加密 TCP 连接 (SSL)
注意	请勿在 <code>espconn</code> 的任何 callback 中调用本接口断开连接。如有需要，可以在 callback 中使用任务触发调用本接口断开连接。
函数定义	<code>sint8 espconn_secure_disconnect(struct espconn *espconn)</code>
参数	<code>struct espconn *espconn</code> : 对应网络连接的结构体
返回	<code>0</code> : 成功 其它: 失败，返回错误码 <code>ESPCONN_ARG</code> : 未找到参数 <code>espconn</code> 对应的 TCP 连接

## 5.8. espconn\_secure\_ca\_enable

功能	开启 SSL CA 认证功能
注意	<ul style="list-style-type: none"><li>CA 认证功能，默认关闭。</li><li>如需调用本接口，请在加密 (SSL) 连接建立前调用：<ul style="list-style-type: none"><li>在 <code>espconn_secure_accept</code> (ESP8266 作为 TCP SSL server) 之前调用；</li><li>或者 <code>espconn_secure_connect</code> (ESP8266 作为 TCP SSL client) 之前调用</li></ul></li></ul>
函数定义	<code>bool espconn_secure_ca_enable (uint8 level, uint32 flash_sector)</code>
参数	<ul style="list-style-type: none"><li><code>uint8 level</code>: 设置 ESP8266 SSL server/client<ul style="list-style-type: none"><li><code>0x01</code>: SSL client</li><li><code>0x02</code>: SSL server</li><li><code>0x03</code>: SSL client 和 SSL server</li></ul></li><li><code>uint32 flash_sector</code>: 设置 CA 证书 <code>esp_ca_cert.bin</code> 烧录到 Flash 的位置。例如，参数传入 <code>0x3B</code>，则对应烧录到 Flash <code>0x3B000</code></li></ul>
返回	<code>true</code> : 成功 <code>false</code> : 失败

## 5.9. espconn\_secure\_ca\_disable

功能	关闭 SSL CA 认证功能
注意	<ul style="list-style-type: none"><li>CA 认证功能，默认关闭。</li><li>如需调用本接口，请在加密 (SSL) 连接建立前调用：<ul style="list-style-type: none"><li>在 <code>espconn_secure_accept</code> (ESP8266 作为 TCP SSL server) 之前调用；</li><li>或者 <code>espconn_secure_connect</code> (ESP8266 作为 TCP SSL client) 之前调用</li></ul></li></ul>



函数定义	<code>bool espconn_secure_ca_disable (uint8 level)</code>
参数	<ul style="list-style-type: none"><li>• <code>uint8 level</code>: 设置 ESP8266 SSL server/client<ul style="list-style-type: none"><li>- <code>0x01</code>: SSL client</li><li>- <code>0x02</code>: SSL server</li><li>- <code>0x03</code>: SSL client 和 SSL server</li></ul></li></ul>
返回	<code>true</code> : 成功 <code>false</code> : 失败

## 5.10. espconn\_secure\_cert\_req\_enable

功能	使能 ESP8266 作为 SSL client 时的证书认证功能
注意	<ul style="list-style-type: none"><li>• 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。</li><li>• 如需调用本接口，请在 <code>espconn_secure_connect</code> 之前调用。</li></ul>
函数定义	<code>bool espconn_secure_cert_req_enable (uint8 level, uint32 flash_sector)</code>
参数	<ul style="list-style-type: none"><li>• <code>uint8 level</code>: 仅支持设置为 <code>0x01</code> ESP8266 作为 SSL client</li><li>• <code>uint32 flash_sector</code>: 设置密钥 <code>esp_cert_private_key.bin</code> 烧录到 Flash 的位置，例如，参数传入 <code>0x3A</code>，则对应烧录到 Flash <code>0x3A000</code>。请注意，不要覆盖到代码或系统参数区域。</li></ul>
返回	<code>true</code> : 成功 <code>false</code> : 失败

## 5.11. espconn\_secure\_cert\_req\_disable

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

## 5.12. espconn\_secure\_set\_default\_certificate

功能	设置 ESP8266 作为 SSL server 时的证书
注意	<ul style="list-style-type: none"><li>• <code>ESP8266_NONOS_SDK/examples/IoT_Demo</code> 中提供使用示例</li><li>• 本接口必须在 <code>espconn_secure_accept</code> 之前调用，传入证书信息</li></ul>



函数定义	<code>bool espconn_secure_set_default_certificate (const uint8_t* certificate, uint16_t length)</code>
参数	<code>const uint8_t* certificate</code> : 证书指针 <code>uint16_t length</code> : 证书长度
返回	<code>true</code> : 成功 <code>false</code> : 失败

### 5.13. espconn\_secure\_set\_default\_private\_key

功能	设置 ESP8266 作为 SSL server 时的密钥
注意	<ul style="list-style-type: none"><li>• <i>ESP8266_NONOS_SDK/examples/IoT_Demo</i> 中提供使用示例</li><li>• 本接口必须在 <code>espconn_secure_accept</code> 之前调用，传入密钥信息</li></ul>
函数定义	<code>bool espconn_secure_set_default_private_key (const uint8_t* key, uint16_t length)</code>
参数	<code>const uint8_t* key</code> : 密钥指针 <code>uint16_t length</code> : 密钥长度
返回	<code>true</code> : 成功 <code>false</code> : 失败



#### 免责声明和版权公告

本文中的信息，包括供参考的 URL 地址，如有变更，恕不另行通知。

文档“按现状”提供，不负任何担保责任，包括对适销性、适用于特定用途或非侵权性的任何担保，和任何提案、规格或样品在他处提到的任何担保。本文档不负任何责任，包括使用本文档内信息产生的侵犯任何专利权行为的责任。本文档在此未以禁止反言或其他方式授予任何知识产权使用许可，不管是明示许可还是暗示许可。

Wi-Fi 联盟成员标志归 Wi-Fi 联盟所有。蓝牙标志是 Bluetooth SIG 的注册商标。

文中提到的所有商标名称、商标和注册商标均属其各自所有者的财产，特此声明。

版权归 © 2017 乐鑫所有。保留所有权利。