



ESP8266 SSL 加密使用手册

Version 1.4

Espressif Systems IOT Team

<http://bbs.espressif.com/>

Copyright © 2016



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。

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© 2016 乐鑫信息科技（上海）有限公司所有。保留所有权利。



目录

1. 前言.....	4
2. ESP8266 作为 SSL server	5
2.1. Linux 环境搭建[可选]	5
2.2. 证书生成.....	7
2.3. 证书使用说明.....	9
3. ESP8266 作为 SSL client	10
3.1. Linux环境搭建[可选]	10
3.2. 证书制作	10
3.3. 证书使用说明[重要].....	14
4. 软件接口	15
4.1. espconn_secure_ca_enable.....	15
4.2. espconn_secure_ca_disable.....	16
4.3. espconn_secure_cert_req_enable	17
4.4. espconn_secure_cert_req_disable	17
4.5. espconn_secure_set_default_certificate.....	18
4.6. espconn_secure_set_default_private_key	18
4.7. espconn_secure_accept.....	19
4.8. espconn_secure_delete	20
4.9. espconn_secure_set_size	20
4.10. espconn_secure_get_size.....	21
4.11. espconn_secure_connect.....	22
4.12. espconn_secure_send	22
4.13. espconn_secure_disconnect	23



1.

前言

SSL 是安全套接层(secure socket layer), TLS 是 SSL 的继任者, 称为传输层安全(transport layer security)。它们共同的作用就是在明文的TCP以上层和TCP层之间加上一层加密层, 这样就保证上层信息传输的安全。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](#)

SSL 通常指代 SSL/TLS 层。

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** 意味着 ESP8266 将把自己的证书传给 SSL client, SSL client 去选择是否校验 ESP8266 证书的合法性。如果是双向认证, 那么 SSL server(ESP8266) 还将要求 SSL client 提供它的证书, 由 SSL server(ESP8266) 决定是否校验 SSL client 的合法性。详见第二章。

ESP8266 作为 **SSL client** 是通常情况, 意味着 ESP8266 将接受 SSL server 传过来的服务器证书, ESP8266 可自由选择是否去校验服务器证书的合法性。如果是双向认证, 那么 SSL client(ESP8266) 还需提供自己的证书给 SSL server, 让 SSL server 选择去校验 SSL client 的合法性。详见第三章。

如您对 **SSL/TLS 认证过程不熟悉, 请仔细理解本文!**

您可结合 <http://blog.csdn.net/ustccw/article/details/76691248> 理解 SSL/TLS 工作过程!



2. ESP8266 作为 SSL server

ESP8266 作为 SSL server 时，用户必须生成 SSL 加密所需的头文件 cert.h 和 private_key.h。如何生成这两个文件，请参考 2.2 节来生成这两个文件。

根据 2.2 节获取到 cert.h 和 private_key.h 之后，用户请参考 IOT_Demo 以及 IOT_Demo 中 `#define SERVER_SSL_ENABLE` 宏定义的代码，实现 SSL server 功能。

CA 认证功能(即双向认证)默认关闭，用户可调用接口 `espconn_secure_ca_enable` 使能 CA 认证。如果调用此接口，必须保证 CA 证书经过 2.2 节的格式转化，并烧录到对应位置。这样，ESP8266 作为 SSL server 时，也需要校验 SSL client 端的证书。

2.1. Linux 环境搭建[可选]

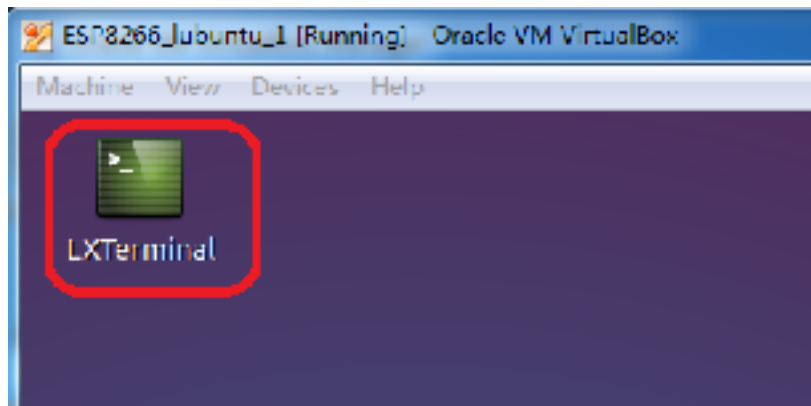
如果用户使用的Linux系统，请跳过2.1节，如果用户使用的是windows系统，请根据本节搭建windows下的Linux环境。

(1) 将证书制作脚本“**makefile.sh**”拷贝到 lubuntu 虚拟机共享路径下。lubuntu 虚拟机编译环境可在 Espressif BBS 下载，

- 下载链接：<http://bbs.espressif.com/viewtopic.php?f=21&t=86>

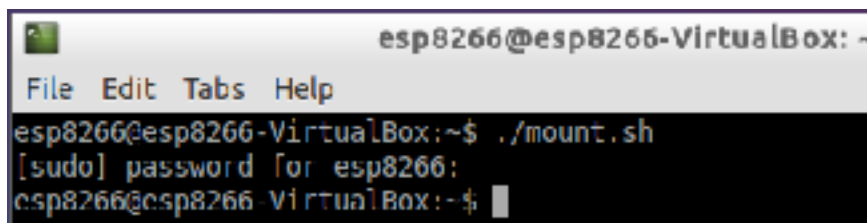
(2) 挂载共享路径。

- 打开虚拟机桌面的“LXTerminal”



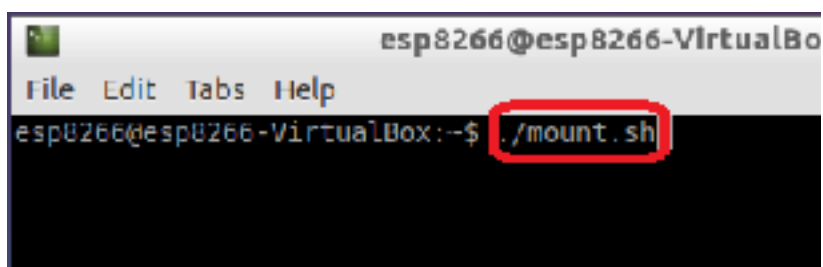


- 输入指令 `./mount.sh`，回车



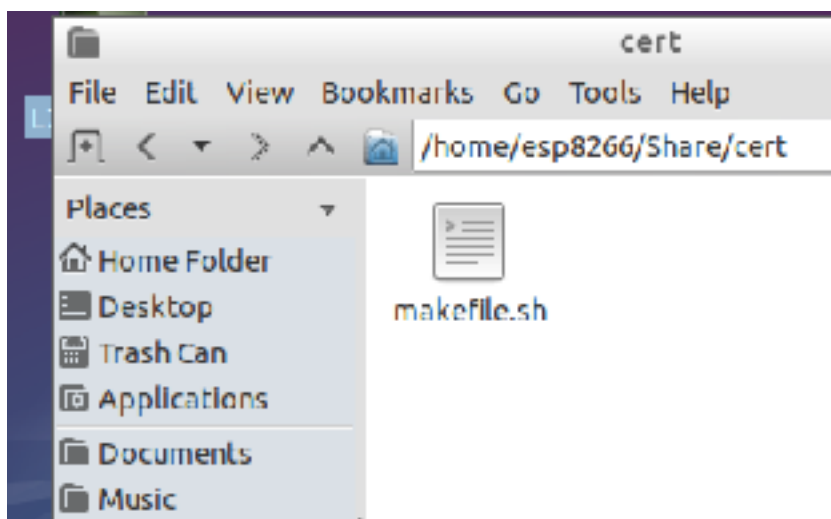
```
esp8266@esp8266-VirtualBox: ~  
File Edit Tabs Help  
esp8266@esp8266-VirtualBox:~$ ./mount.sh  
[sudo] password for esp8266:  
esp8266@esp8266-VirtualBox:~$
```

- 输入密码 **espressif**，回车



```
esp8266@esp8266-VirtualBo  
File Edit Tabs Help  
esp8266@esp8266-VirtualBox:~$ ./mount.sh
```

- 在虚拟机打开共享文件夹，看到证书制作脚本，挂载成功。



- 挂载成功后，请依次拷贝 **make_cacert.py**, **make_cert.py**, **refile.sh** 到 `makefile.sh` 同目录下。

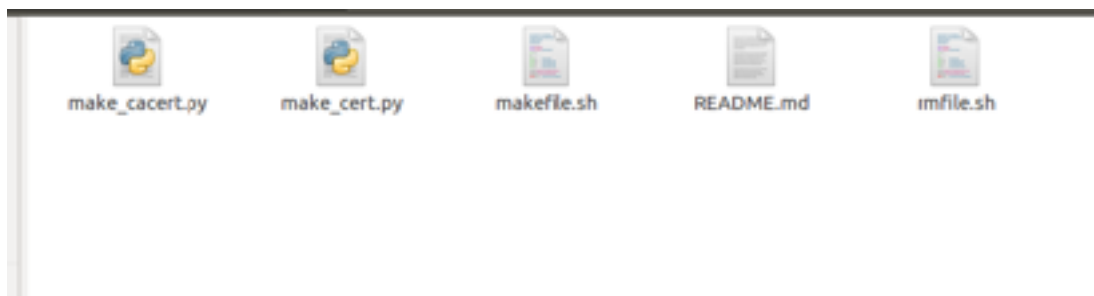


2.2. 证书生成

请根据您的情况选择下面 (a) (b) 中其一来生成 SSL 加密所需的头文件 cert.h 和 private_key.h。和 esp_ca_cert.bin [仅双向认证需要烧录]。

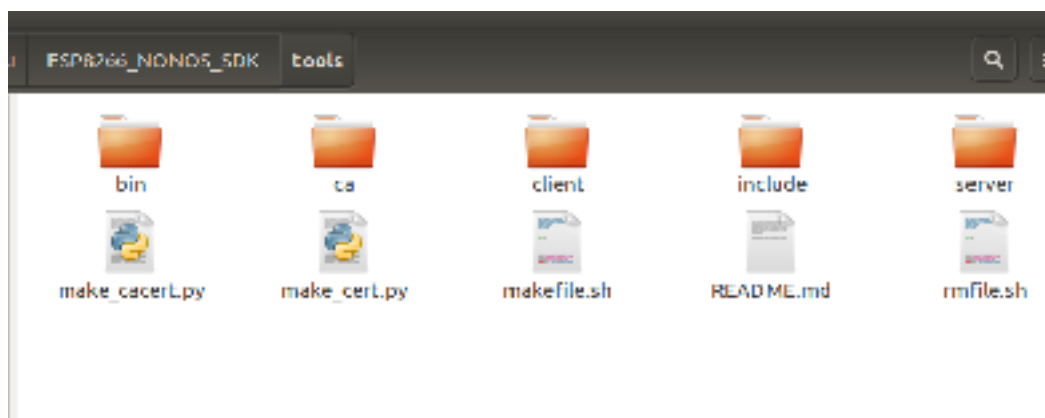
(a) 如果您没有正式的 CA 机构颁发的证书

我们提供了如图工具，将生成自签 CA (自己作为 CA，仅测试使用)，同时使用自签 CA 给自己颁发服务器证书。



用户需修改 **makefile.sh** 中的 **CN** 字段，由 192.168.111.100 改为你自己主机的 IP 地址。然后直接执行命令 **makefile.sh** 即可自动生成所有相关加密文件。

```
$/makefile.sh
```



所生成的 **SSL** 加密所需的头文件 **cert.h** 和 **private_key.h** 位于图中 **include** 目录下。双向认证所需的 **esp_ca_cert.bin** 位于 **bin** 目录下。



其他说明：

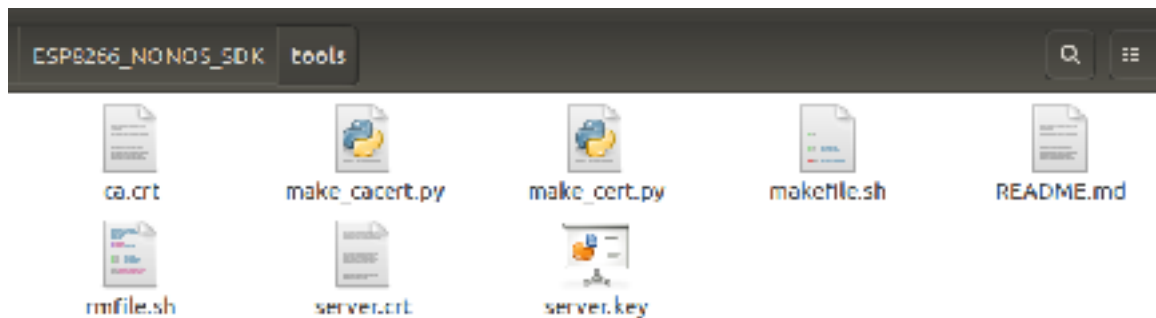
- 用户可根据需要加密需要，将 makefile.sh 中默认的 1024 位加密改为 512 位或其他
- rmfile.sh 可删除产生过的所有文件
- ca 目录为自签的 CA
- make_cacert.py 和 make_cert.py 为证书格式转化和生成用到的工具

(b) 如果您有私钥 (server.key) , 并且有正式的 CA 机构 (ca.crt) 颁发的证书 (server.crt)

请将您的 server.key, ca.crt, server.crt 拷贝至 ESP8266/tools/目录下。如图所示：

说明：

- [重要] 如果不是此名，请一定要对应重命名为 server.key, ca.crt, server.crt
- 请确保 ca.crt 和 server.crt 为 PEM 格式

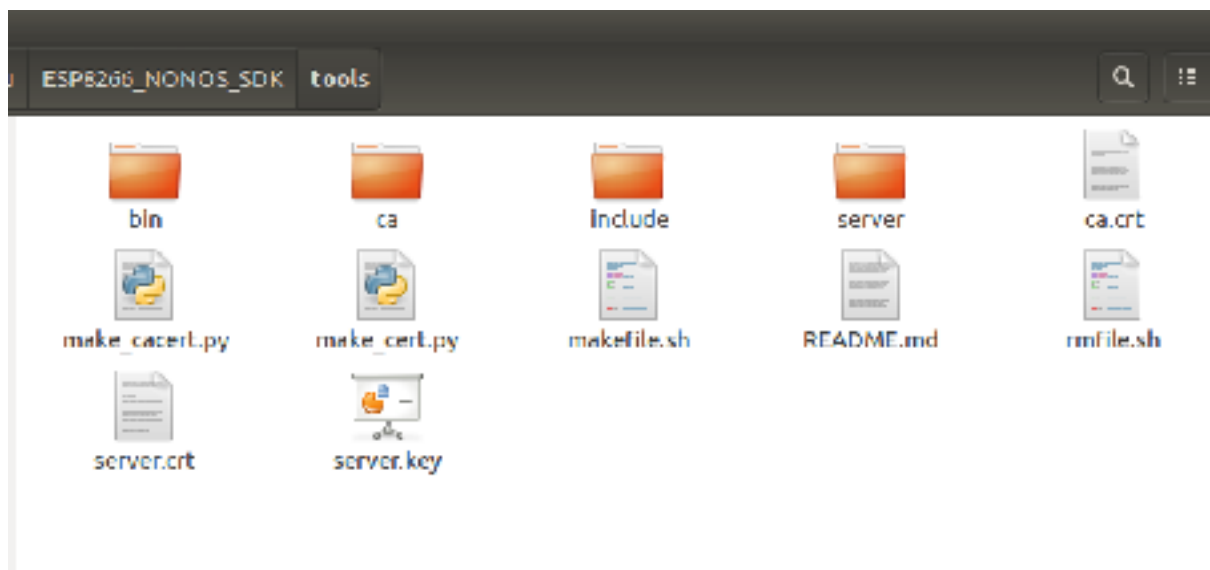


然后直接执行命令 makefile.sh 即可自动生成所有相关加密文件。

```
$/makefile.sh
```

所生成的 SSL 加密所需的头文件 **cert.h** 和 **private_key.h** 位于下图中 **include** 目录下。

双向认证所需的 **esp_ca_cert.bin** 位于 **bin** 目录下。



其他说明：

- 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

2.3. 证书使用说明[重要]

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。

- 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**
- 开发者如需双向认证，必须调用 espconn_secure_ca_enable 指定证书位置,见 4.1 节说明
- 开发者如需双向认证, 必须烧录 CA 证书 esp_ca_cert.bin 到 espconn_secure_ca_enable 指定的位置
- [重要] 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，**如内存不足，会导致 SSL handshake 失败**
- [重要] 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。**如内存不足，会导致 SSL handshake 失败**



3. ESP8266 作为 SSL client

ESP8266 作为 SSL client 时，用户可根据需要生成 SSL 加密所需的证书文件。

单向认证[即ESP8266校验服务器合法性]：需生成 ca 证书文件 `esp_ca_cert.bin`，参考 3.2 节。

双向认证：需生成 ca 证书文件 `esp_ca_cert.bin` 以及 client 证书私钥文件 `esp_cert_private_key.bin`。如何生成这两个文件，请参考 3.2 节来生成这两个文件。

根据 3.2 节获取到 `esp_ca_cert.bin/esp_cert_private_key.bin` 之后，用户请参考 **esp_mqtt_demo** 以及 `esp_mqtt_demo` 中 `#define MQTT_SSL_ENABLE` 宏定义的代码，实现 SSL client 功能。

单向认证功能(即ESP8266校验服务器)默认关闭，用户可调用接口 `espconn_secure_ca_enable` 打开单向认证。如果调用此接口，必须保证 CA 证书经过3.2节的格式转化，转化为 `esp_ca_cert.bin`，并烧录到对应位置。这样，ESP8266 作为 SSL client 时，将校验 SSL server 端的证书。

双向认证功能(单向认证基础上，将自己证书也传给 SSL server)默认关闭，用户可调用 `espconn_secure_cert_req_enable` 打开双向认证。如果调用此接口，必须保证 client 证书私钥经过 3.2 节的格式转化，转化为 `esp_cert_private_key.bin`，并烧录到对应位置。这样，ESP8266 作为 SSL client 时，不仅校验服务器证书，同时也将自己的证书传给 SSL server。

3.1. Linux环境搭建[可选]

如果用户使用的Linux系统，请跳过3.1节，如果用户使用的是windows系统，请根据 2.1 节 搭建windows下的Linux环境。

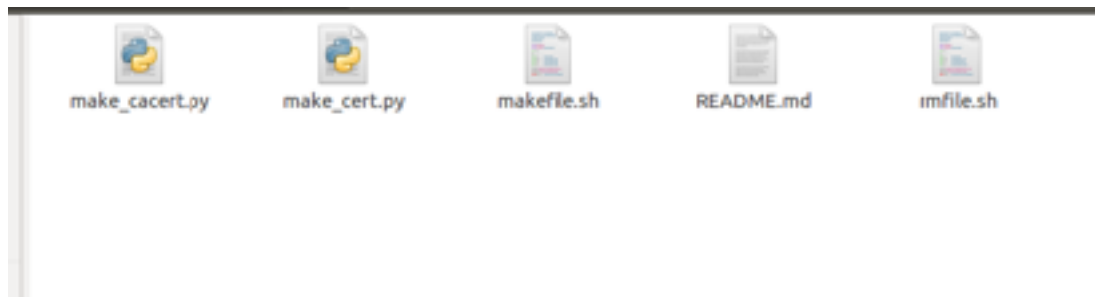
3.2. 证书制作

请根据您的情况选择下面 (a)(b)(c)中其一生成 SSL 加密中, 单向认证所需的 `esp_ca_cert.bin` 或双向认证所需的 `esp_ca_cert.bin` 和 `esp_cert_private_key.bin`



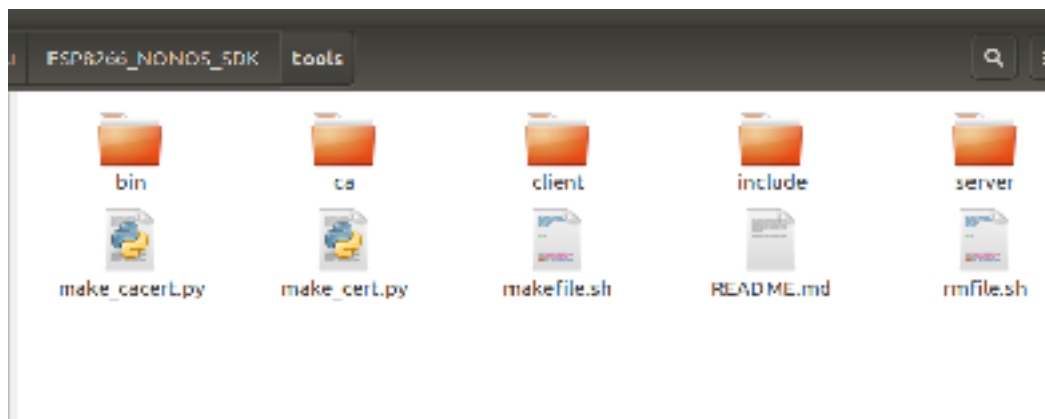
(a) 如果您没有任何正式的 CA 机构颁发的证书

在 tools 目录下，我们提供如图工具，该工具将生成自签的 CA(ca.crt + ca.key), 同时使用自签的 CA 给自己颁发证书。



用户需修改 **makefile.sh** 中的 **CN** 字段，由 192.168.111.100 改为你自己主机的 IP 地址。
然后直接执行命令 **makefile.sh** 即可自动生成所有相关加密文件。

```
$/makefile.sh
```



所生成的 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：

- 用户可根据需要加密需要，将 **makefile.sh** 中默认的 1024 位加密改为 512 位或其他
- **rmfile.sh** 可删除产生过的所有文件
- **ca** 目录为自签的 CA
- **make_cacert.py** 和 **make_cert.py** 为证书格式转化和生成用到的工具

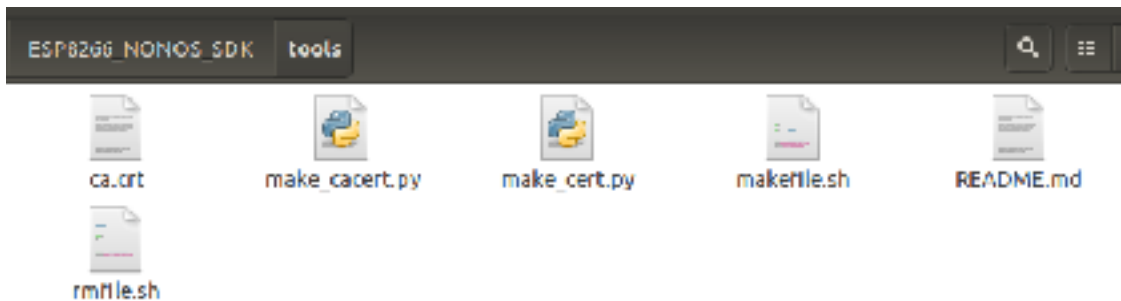


(b) 如果您仅仅有正式的 CA 机构的证书 **ca.crt**

将 **ca.crt** 拷贝至 **tools** 目录下，如图：

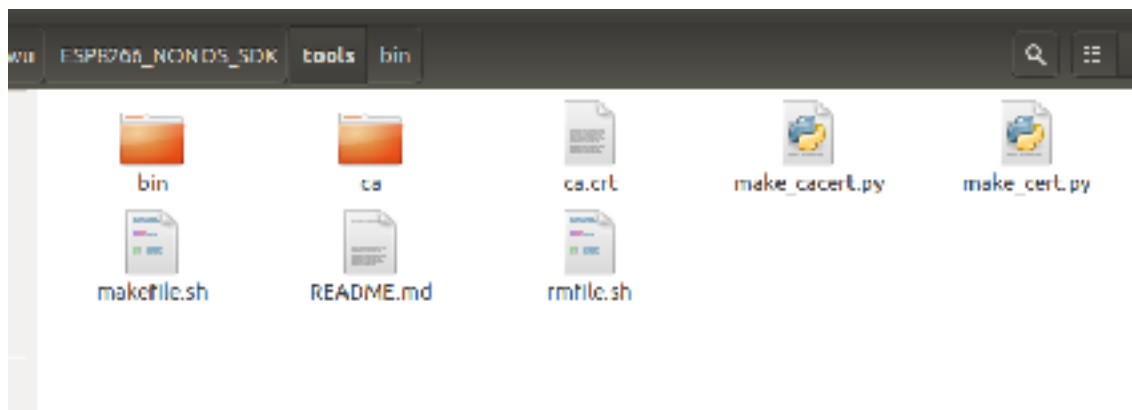
说明：

- [重要] 如果不是此名，请一定要对应重命名为 **ca.crt**
- 请确保 **ca.crt** 为 PEM 格式



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

`$/makefile.sh`



单向认证所需的 **esp_ca_cert.bin** 文件位于 **bin** 目录下。

其他说明：

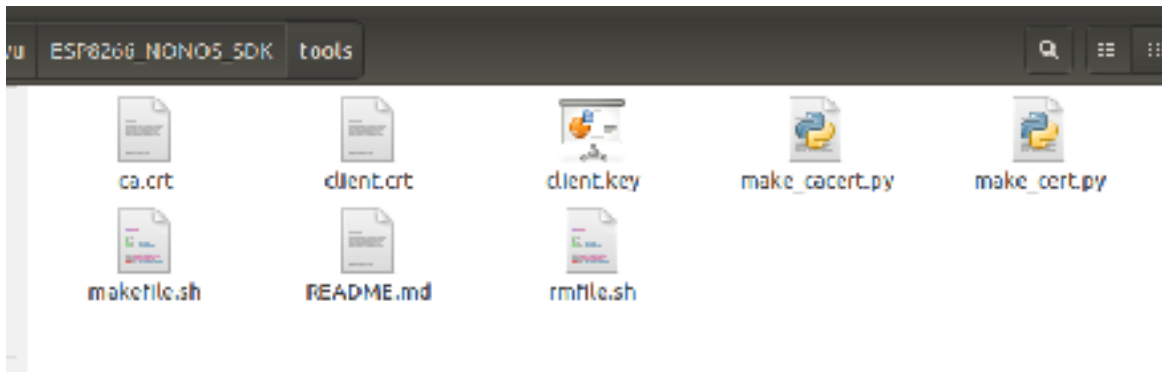
- **rmfile.sh** 可删除产生过的所有文件
- **make_cacert.py** 和 **make_cert.py** 为证书格式转化和生成用到的工具



(c) 如果您有正式的 **CA** 机构的证书 **ca.crt**，并且有私钥 **client.key** 以及由该 **CA** 颁发的证书 **client.crt** 将 ca.crt, client.key, client.crt 拷贝至 tools 目录下，如图：

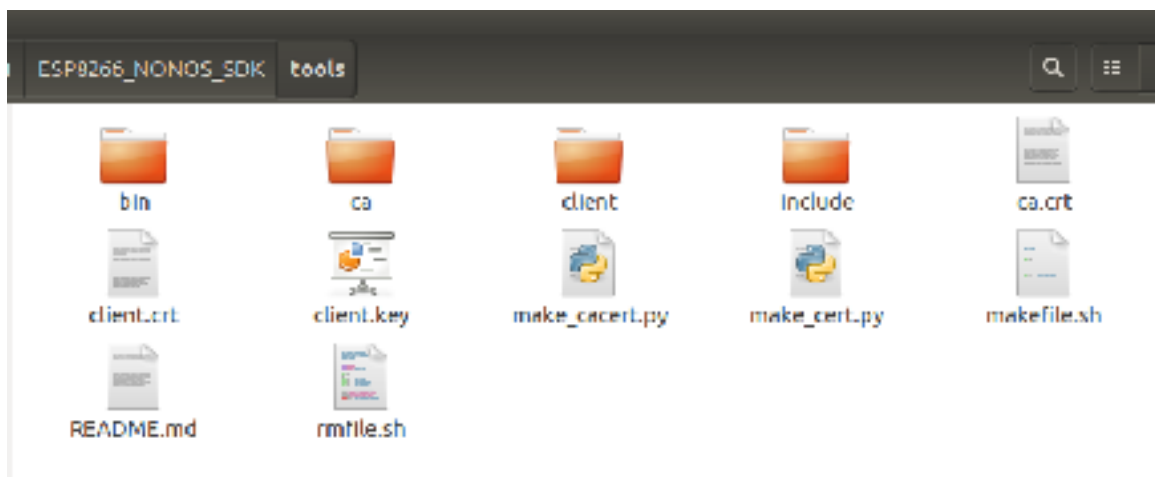
说明：

- [重要] 如果不是此名，请一定要对应重命名为 ca.crt, client.key, client.crt
- 请确保 ca.crt, client.crt 为 PEM 格式



然后直接执行命令 makefile.sh 即可自动生成所有相关加密文件。

```
$/makefile.sh
```



所生成的 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。



3.3. 证书使用说明[重要]

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

- 如果 ESP8266 需校验服务器证书，必须调用接口 `espconn_secure_ca_enable` 来打开认证
- 如果 ESP8266 需校验服务器证书，必须通过烧写工具烧录 `esp_ca_cert.bin`, 烧录的位置由 `espconn_secure_ca_enable` 第二个参数决定，详见 4.1 节
- 如果 SSL server 需校验 ESP8266 的证书，必须调用接口 `espconn_secure_cert_req_enable`
- 如果 SSL server 需校验 ESP8266 的证书，必须通过烧写工具烧录 `esp_cert_private_key.bin`, 烧录的位置由 `espconn_secure_cert_req_enable` 第二个参数决定，详见 4.3 节
- 如果是双向认证, 需调用 `espconn_secure_ca_enable` 和 `espconn_secure_cert_req_enable`
- 如果是双向认证, 必须通过烧写工具烧录 `esp_ca_cert.bin` 和 `esp_cert_private_key.bin`, 烧写位置由 `espconn_secure_ca_enable` 和 `espconn_secure_cert_req_enable` 第二个参数决定
- [重要] 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，**如内存不足，会导致 SSL handshake 失败**
- [重要] 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。**如内存不足，会导致 SSL handshake 失败**



4.

软件接口

SSL 系列软件接口与普通 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 编程手册“2C-ESP8266__SDK__API Guide”

SSL 连接，用户可参考 BBS 提供的 Demo <http://bbs.espressif.com/viewtopic.php?f=21&t=389>

4.1. `espconn_secure_ca_enable`

功能：

开启 SSL CA 认证功能

注意：

- CA 认证功能，默认关闭
- 如需调用本接口，必须烧录 `esp_ca_cert.bin`
- 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

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 `0x3B`, 则对应烧录到 Flash `0x3B000`。请注意, 不要覆盖了代码或系统参数区域。

返回:

true : 成功

false : 失败

4.2. `espconn_secure_ca_disable`

功能:

关闭 SSL CA 认证功能

注意:

- CA 认证功能, 默认关闭

函数定义:

`bool espconn_secure_ca_disable (uint8 level)`

参数:

`uint8 level` : 设置 ESP8266 SSL server/client:

`0x01` SSL client;

`0x02` SSL server;

`0x03` SSL client 和 SSL server

返回:

true : 成功

false : 失败



4.3. `espconn_secure_cert_req_enable`

功能：

使能 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。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。

函数定义：

```
bool espconn_secure_cert_req_enable (uint8 level, uint32  
flash_sector)
```

参数：

`uint8 level` : 仅支持设置为 `0x01` ESP8266 作为 SSL client;

`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 `0x3A`，则对应烧录到 Flash `0x3A000`。请注意，不要覆盖了代码或系统参数区域。

返回：

`true` : 成功
`false` : 失败

4.4. `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：

```
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

4.5. `espconn_secure_set_default_certificate`

功能:

设置 ESP8266 作为 SSL server 时的证书

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

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

4.6. `espconn_secure_set_default_private_key`

功能:

设置 ESP8266 作为 SSL server 时的密钥

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

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

4.7. `espconn_secure_accept`

功能：

创建 SSL TCP server，侦听 SSL 握手。

注意：

- 本接口只能调用一次，仅支持建立一个 SSL server，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，ESP8266 无法处理，SSL 连接断开，进入 `espconn_reconnect_callback`
- 如需创建 SSL server，需先调用 `espconn_secure_set_default_certificate` 和 `espconn_secure_set_default_private_key` 传入证书和密钥。

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
- Non-0 : 失败, 返回错误码
 - ESPCONN_ARG - 未找到参数 `espconn` 对应的 TCP 连接
 - ESPCONN_MEM - 空间不足
 - ESPCONN_ISCONN - 连接已经建立

4.8. `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` : 对应网络传输的结构体

返回:

- 0 : 成功
- Non-0 : 失败, 返回错误码
 - ESPCONN_ARG - 未找到参数 `espconn` 对应的网络传输
 - 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。

4.9. `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 2KBytes; 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 `espconn_secure_connect` (ESP8266 作为 TCP SSL client) 之前调用

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 ~ 8192, 单位: 字节

默认值为 `2048`

返回:

`true` : 成功

`false` : 失败

4.10. `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 ESP8266 SSL server/client:

`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



4.11. `espconn_secure_connect`

功能：

加密（SSL）连接到 TCP SSL server（ESP8266 作为 TCP SSL client）

注意：

- 目前 ESP8266 作为 SSL client 仅支持一个连接，本接口如需多次调用，请先调用 `espconn_secure_disconnect` 断开前一次连接，再建立下一个 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`

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

Non-0 : 失败，返回错误码

`ESPCONN_ARG` - 未找到参数 `espconn` 对应的 TCP 连接

`ESPCONN_MEM` - 空间不足

`ESPCONN_ISCONN` - 传输已经建立

4.12. `espconn_secure_send`

功能：

发送加密数据（SSL）

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。



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
Non-0 : 失败, 返回错误码 `ESPCONN_ARG` - 未找到参数 `espconn` 对应的 TCP 连接

4.13. 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
Non-0 : 失败, 返回错误码 `ESPCONN_ARG` - 未找到参数 `espconn` 对应的 TCP 连接