



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. 证书制作	5
3. ESP8266 作为 SSL client	9
3.1. 证书制作	9
4. 软件接口	10
4.1. espconn_secure_ca_enable.....	10
4.2. espconn_secure_ca_disable.....	11
4.3. espconn_secure_cert_req_enable	12
4.4. espconn_secure_cert_req_disable	12
4.5. espconn_secure_set_default_certificate.....	13
4.6. espconn_secure_set_default_private_key	13
4.7. espconn_secure_accept	14
4.8. espconn_secure_delete	15
4.9. espconn_secure_set_size	15
4.10. espconn_secure_get_size.....	16
4.11. espconn_secure_connect	17
4.12. espconn_secure_send	17
4.13. espconn_secure_disconnect	18



1.

前言

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。

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。

SSL 证书生成脚本及示例：[test_cert_and_creat_information](#) 和 [TLS_BiDirectVerif_Demo](#)。



2. ESP8266 作为 SSL server

ESP8266 作为 SSL server 时，提供加密证书的制作脚本，生成 SSL 加密所需的头文件 cert.h 和 private_key.h。用户可以参考 IOT_Demo 中 `#define SERVER_SSL_ENABLE` 宏定义的代码，实现 SSL server 功能。

CA 认证功能默认关闭，用户可调用接口 `espconn_secure_ca_enable` 使能 CA 认证。

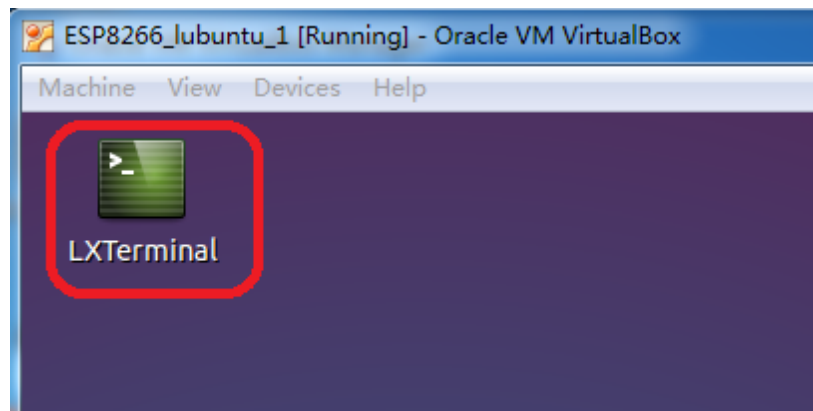
2.1. 证书制作

(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>

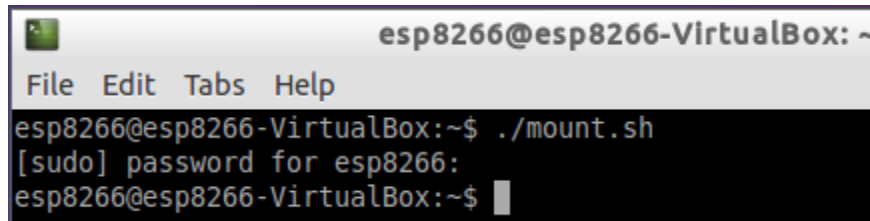
(2) 挂载共享路径。

- 打开虚拟机桌面的 “LXTerminal”



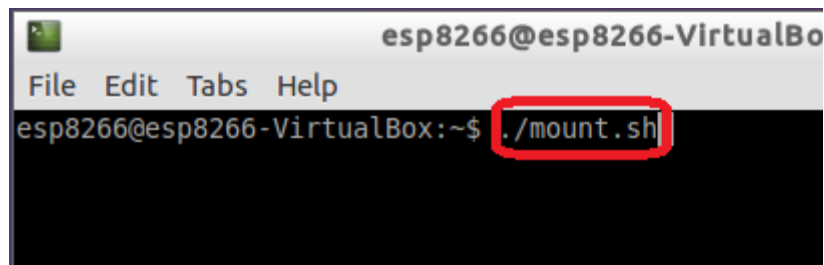


- 输入指令 `./mount.sh` , 回车



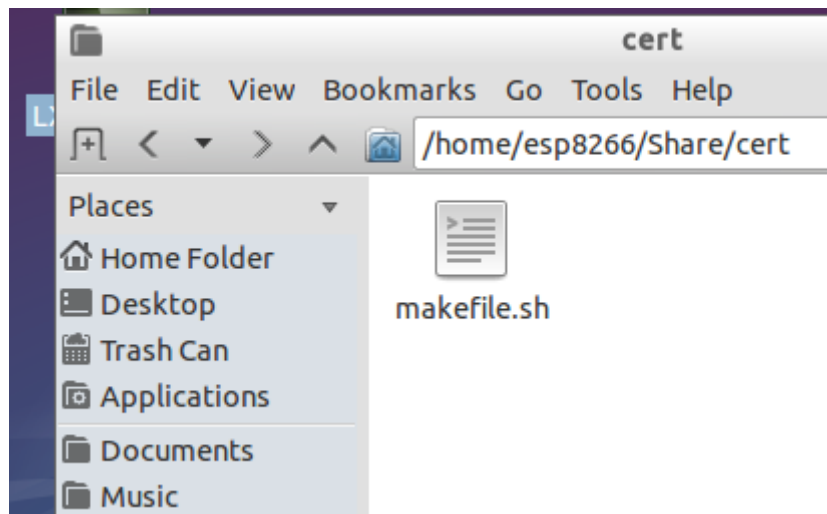
```
esp8266@esp8266-VirtualBox: ~  
File Edit Tabs Help  
esp8266@esp8266-VirtualBox:~$ ./mount.sh  
[sudo] password for esp8266:  
esp8266@esp8266-VirtualBox:~$
```

- 输入密码 **espressif** , 回车



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

(3) 在虚拟机打开共享文件夹，看到证书制作脚本，挂载成功。

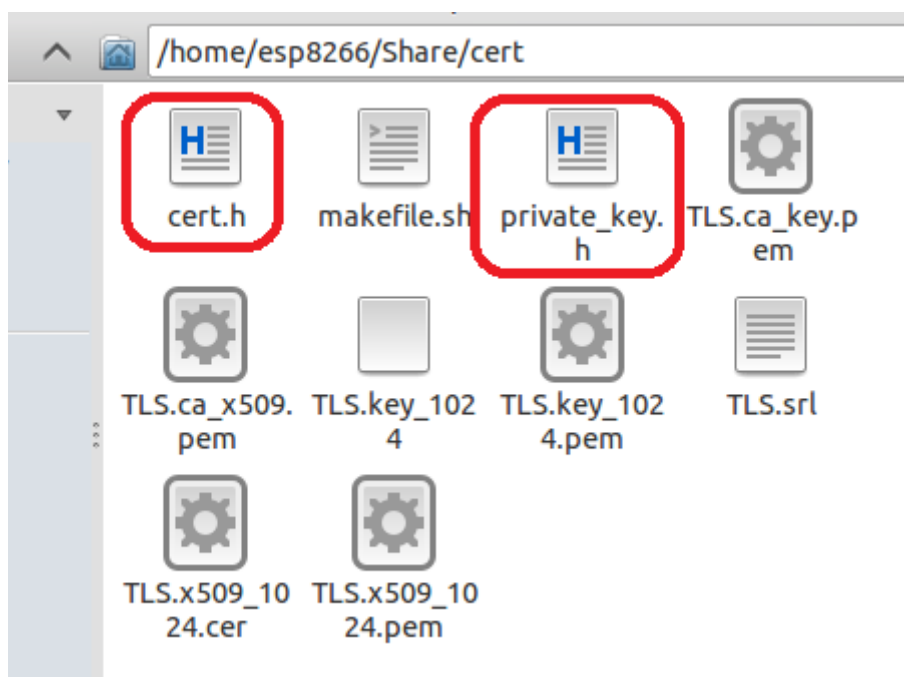




- (4) 在脚本所在路径，运行脚本，指令 `./makefile.sh`，生成 `cert.h` 和 `private_key.h`，参考 IOT_Demo 使用这两个头文件即可。

```
esp8266@esp8266-VirtualBox:~$ cd /home/esp8266/Share/cert
esp8266@esp8266-VirtualBox:~/Share/cert$ ./makefile.sh
```

运行成功，生成如下：



注意事项

- 证书制作脚本 `makefile.sh` 文件中的 IP 地址，请改为用户实际的 server IP，如下图：

```
cat > certs.conf << EOF
[ req ]
distinguished_name = req_distinguished_name
prompt             = no

[ req_distinguished_name ]
0                   = $PROJECT_NAME
CN                  = 127.0.0.1
EOF
```



- 证书制作脚本 makefile.sh 默认为 1024 的加密算法位数长度；如需使用 512 的加密算法位数，

```
# private key generation
openssl genrsa -out TLS.ca_key.pem 1024
openssl genrsa -out TLS.key_1024.pem 1024

# convert private keys into DER format
openssl rsa -in TLS.key_1024.pem -out TLS.key_1024 -outform DER

# cert requests
openssl req -out TLS.ca_x509.req -key TLS.ca_key.pem -new \
    -config ./ca_cert.conf
openssl req -out TLS.x509_1024.req -key TLS.key_1024.pem -new \
    -config ./certs.conf

# generate the actual certs.
openssl x509 -req -in TLS.ca_x509.req -out TLS.ca_x509.pem \
    -sha1 -days 5000 -signkey TLS.ca_key.pem
openssl x509 -req -in TLS.x509_1024.req -out TLS.x509_1024.pem \
    -sha1 -CAcreateserial -days 5000 \
    -CA TLS.ca_x509.pem -CAkey TLS.ca_key.pem

# some cleanup
rm TLS*.req
rm *.conf

openssl x509 -in TLS.ca_x509.pem -outform DER -out TLS.ca_x509.cer
openssl x509 -in TLS.x509_1024.pem -outform DER -out TLS.x509_1024.cer[]

#
# Generate the certificates and keys for encrypt.
#
```

请修改 makefile.sh 文件中的 1024 改为 512：

注意：

- 从 ESP8266_NONOS_SDK_V1.4.0 起，开发者必须调用 `espconn_secure_set_default_certificate` 和 `espconn_secure_set_default_private_key` 传入证书和密钥。
- 证书制作脚本 makefile.sh 生成的默认 SSL server 证书由 Espressif Systems 颁发，并非由 CA 颁发。如果用户需要 CA 认证，请将运行脚本 makefile.sh 生成的 TLS.ca_x509.cer 导入 SSL client，并参考后文 **3.1 证书制作** 使用脚本 “`make_cacert.py`” 将 CA 文件生成 esp_ca_cert.bin 烧写到 Flash 对应的地址。



3. ESP8266 作为 SSL client

开发者可以参考 IOT_Demo 中 `#define CLIENT_SSL_ENABLE` 宏定义的代码，实现 SSL client 功能。ESP8266 作为 SSL client 时，可支持双向认证。

CA 认证功能默认关闭，开发者可调用接口 `espconn_secure_ca_enable` 使能 CA 认证。

ESP8266 作为 SSL client 时支持证书认证功能，但此功能默认关闭，开发者可以调用接口 `espconn_secure_cert_req_enable` 使能证书认证。

开发者可以参考 SSL 证书示例 “TLS_BiDirectVerif_Demo”。

3.1. 证书制作

- (1) 修改脚本 `makefile.sh`，制作开发者自行签发的 CA 证书，例如，证书示例中的 `TLS.ca_x509.cer`。
- (2) 使用签发的 CA 制作供 SSL client 使用的证书，例如，证书示例中的 `TLS.x509_1024.cer`。
- (3) 取出制作 SSL client 使用的证书时所用的密钥，例如，证书示例中的 `TLS.key_1024`。
- (4) 将证书合成脚本 “`make_cacert.py`” 与 CA 文件（例如 `TLS.ca_x509.cer`）放在同一目录下。
- (5) 运行脚本 “`make_cacert.py`” 将合成同一目录下的 CA 文件生成 `esp_ca_cert.bin`，`esp_ca_cert.bin` 的烧录位置由接口 `espconn_secure_ca_enable` 设置，用户可自行定义。
- (6) 重命名证书名称（例如 `TLS.x509_1024.cer`）改为 `certificate.cer`；重命名密钥名称（例如 `TLS.key_1024`）改为 `private_key.key_1024`。请注意，此步骤中必须重新命名，否则将导致认证失败。
- (7) 将重命名后的文件，与脚本 `make_cert.py` 拷贝到同一目录下。
- (8) 运行脚本 `make_cert.py` 生成 `esp_cert_private_key.bin`，`esp_cert_private_key.bin` 的烧录位置由接口 `espconn_secure_cert_req_enable` 设置，用户可自行定义。



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_XXX` 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