ESP8266 SSL User Manual

Version 1.1

Espressif Systems IOT Team

Disclaimer and Copyright Notice

Information in this document, including URL references, is subject to change without notice.

THIS DOCUMENT IS PROVIDED AS IS WITH NO WARRANTIES WHATSOEVER, INCLUDING ANY WARRANTY OF MERCHANTABILITY, NON-INFRINGEMENT, FITNESS FOR ANY PARTICULAR PURPOSE, OR ANY WARRANTY OTHERWISE ARISING OUT OF ANY PROPOSAL, SPECIFICATION OR SAMPLE. All liability, including liability for infringement of any proprietary rights, relating to use of information in this document is disclaimed. No licenses express or implied, by estoppel or otherwise, to any intellectual property rights are granted herein.

The WiFi Alliance Member Logo is a trademark of the WiFi Alliance.

All trade names, trademarks and registered trademarks mentioned in this document are property of their respective owners, and are hereby acknowledged.

Table of Contents

1. Preambles

Herein we introduce ESP8266 SDK SSL user manual, includes that ESP8266 runs as SSL server and ESP8266 runs as SSL client.

SSL function requires a lot of RAM memory, users need to make sure that there is enough space before running the application. If SSL buffer is set to be 8KB by `espconn_secure_set_size`, SSL function requires 22KB at least, it depends on the size of certificate from SSL server.

More information about ESP8266 is on BBS：http://bbs.espressif.com/

2. ESP8266 as SSL server

Sample code of ESP8266 running as SSL server is in IOT_Demo marked with `#define SERVER_SSL_ENABLE`. Espressif Systems offers a script "**makefile.sh**" to generate the ".h" header files which are needed when ESP8266 running as SSL server.

CA verify function default to be disabled, user can enable it by `espconn_secure_ca_enable`.

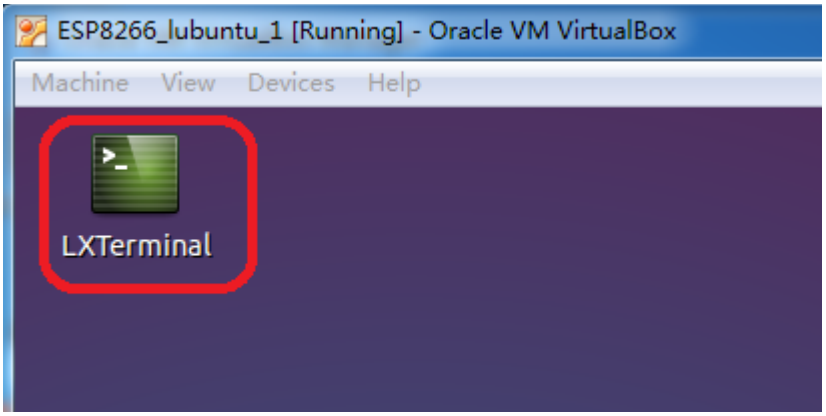
2.1. Generate certificate

(1) Copy script "makefile.sh" to the shared folder of virtual box lubuntu.

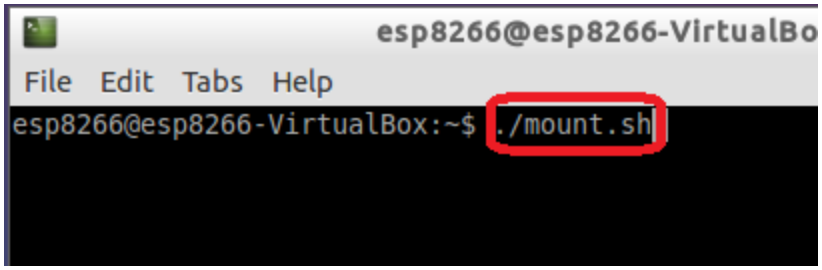
- How to set up the lubuntu compile environment, please refer to BBS： http://bbs.espressif.com/viewtopic.php?f=21&t=86

(2) Mount the shared folder

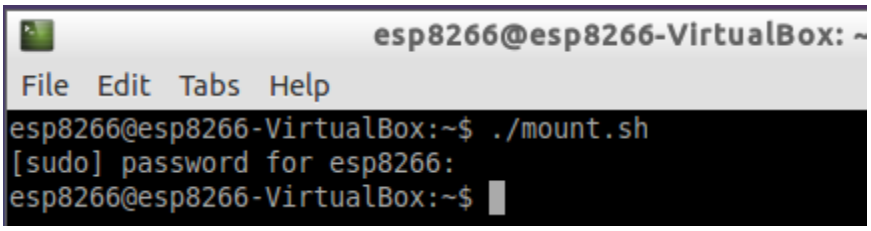
- Open "LXTerminal" in virtual box



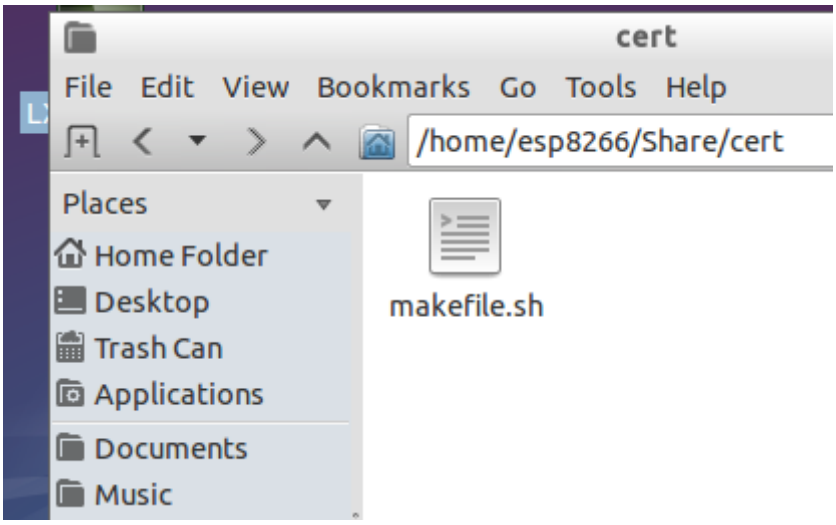
- input command `./mount.sh`



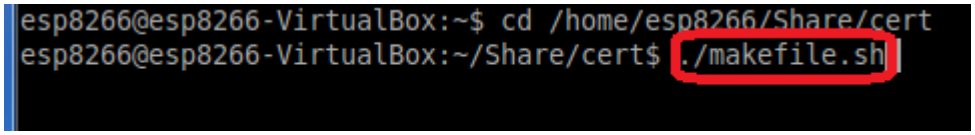
- input password: **espressif**



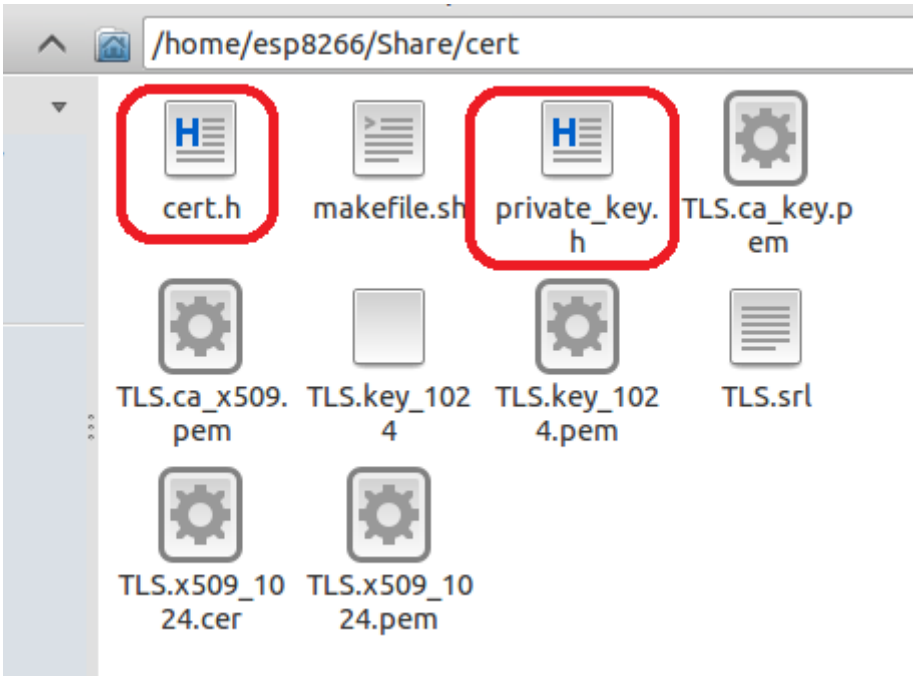
(3) Open shared folder in virtual box, and get script "makefile.sh" there.



(4) Input command `./makefile.sh` to run script "makefile.sh" there.



Generate `cert.h` and `private_key.h`, using these 2 header files according to IOT_Demo:



Notice:

- IP address in script "makefile.sh" need to be user's actual SSL server IP

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

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

- Script "makefile.sh" default to use 1024bit encryption algorithm, if user needs to use 512bit encryption algorithm, please revise script "makefile.sh", change the 1024 to 512.

```
# 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.
#
```

- Certificates generated above is issued by Espressif Systems, not CA. So if users need CA verify, there are 2 methods :
  - Add `TLS.ca_x509.cer` which generated as above into SSL client's trust anchor, then generate `esp_ca_cert.bin` by script "**make_cert.py**" according to **3.1 Generate CA Certificate**, and download `esp_ca_cert.bin` into flash
  - Using CA certificate to generate `cert.h` and `private_key.h`, this needs user to revise script "**makefile.sh**" themselves. Then generate `esp_ca_cert.bin` by script "**make_cert.py**" according to **3.1 Generate CA Certificate**, and download `esp_ca_cert.bin` into flash

3. ESP8266 as SSL client

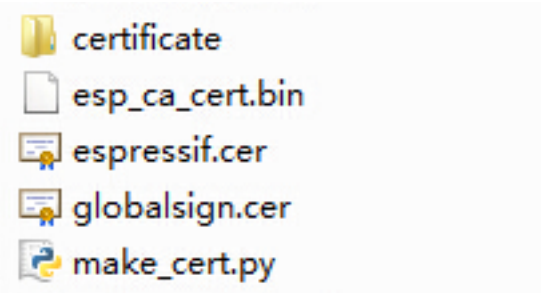
Sample code of ESP8266 running as SSL client is in IOT_Demo marked with `#define CLIENT_SSL_ENABLE`. Espressif Systems offers a script "**make_cert.py**" to generate CA certificate.

CA verify function default to be disabled, user can enable it by `espconn_secure_ca_enable`.

3.1. Generate CA Certificate

(1) Put script "make_cert.py" and CA certificate into the same folder.

(2) Run script "make_cert.py" to generate `esp_ca_cert.bin` which contains all CA certificates (2 CA certificates at most) in the same folder. Download address of `esp_ca_cert.bin` depends on `espconn_secure_ca_enable`.



3.2. CA Verify

STEP 1: ESP8266 connects to server, read `esp_ca_cert.bin` from flash, get the corresponding SSL ctx. Only 2 CA certificates is allowed at most.

STEP 2: ESP8266 starts TLS handshake, get certificate from SSL server, check with the CA in step 1:

- if CA check fail, connection break;
- if succeed, CA verify pass.

4. Software APIs

SSL related APIs are different from normal TCP APIs, so please don't mixed use. In SSL connection, only APIs below can be called:

- espconn_secure_XXX APIs which are SSL related APIs
- `espconn_regist_XXX` APIs to register callbacks
- `espconn_port` to get an available port

Herein we only introduce `espconn_secure_XXX` APIs, more details about software APIs, please refer to documentation "2C-ESP8266__SDK__Programming Guide"

Here is a demo of SSL connection on BBS http://bbs.espressif.com/viewtopic.php?f=21&t=389

4.1. espconn_secure_ca_disable

Function:

Disable SSL CA (certificate authenticate) function

Note:

- CA function is disabled by default,
- If user want to call this API, please call it before
  `espconn_secure_accept` (ESP8266 as TCP SSL server) or
  `espconn_secure_connect` (ESP8266 as TCP SSL client)

Prototype:

```
bool espconn_secure_ca_disable (uint8 level)
```

Parameter:

`uint8 level` : set configuration for ESP8266 SSL server/client:

- `0x01` SSL client;
- `0x02` SSL server;
- `0x03` both SSL client and SSL server

Return:

true : succeed
false : fail

4.2. espconn_secure_ca_enable

Function:

Enable SSL CA (certificate authenticate) function

Note:

- CA function is disabled by default
- If user want to call this API, please call it before espconn_secure_accept (ESP8266 as TCP SSL server) or espconn_secure_connect (ESP8266 as TCP SSL client)

Prototype:

```
bool espconn_secure_ca_enable (uint8 level, uint16 flash_sector)
```

Parameter:

uint8 level : set configuration for ESP8266 SSL server/client:

0x01 SSL client;

0x02 SSL server;

0x03 both SSL client and SSL server

uint16 flash_sector : flash sector in which CA (esp_ca_cert.bin) is downloaded. For example, flash_sector is 0x3B, then esp_ca_cert.bin need to download into flash 0x3B000

Return:

true : succeed

false : fail

4.3. espconn_secure_accept

Function:

Creates an SSL TCP server.

Note:

- Only created one SSL server is allowed, this API can be called only once, and only one SSL client is allowed to connect.

- If SSL encrypted packet size is larger than ESP8266 SSL buffer size (default 2KB, set by espconn_secure_set_size), SSL connection will fail, will enter espconn_reconnect_callback

Prototype:

sint8 espconn_secure_accept(struct espconn *espconn)

Parameter:

struct espconn *espconn : corresponding connected control block structure

Return:

0 : succeed

Non-0 : error code

ESPCONN_MEM – Out of memory

ESPCONN_ISCONN – Already connected

ESPCONN_ARG – illegal argument, can't find TCP connection according to structure espconn

4.4. espconn_secure_set_size

Function:

Set buffer size of encrypted data (SSL)

Note:

Buffer size default to be 2Kbytes. If need to change, please call this API before espconn_secure_accept (ESP8266 as TCP SSL server) or espconn_secure_connect (ESP8266 as TCP SSL client)

Prototype:

bool espconn_secure_set_size (uint8 level, uint16 size)

Parameters:

uint8 level : set buffer for ESP8266 SSL server/client:

0x01 SSL client;

0x02 SSL server;

```
                    0x03  both SSL client and SSL server
    uint16 size : buffer size, range: 1 ~ 8192, unit: byte, default is
    2048

Return:
    true   : succeed
    false  : fail
```

4.5. espconn_secure_get_size

```
Function:
    Get buffer size of encrypted data (SSL)

Prototype:
    sint16 espconn_secure_get_size (uint8 level)

Parameters:
    uint8 level : buffer for ESP8266 SSL server/client:
                    0x01  SSL client;
                    0x02  SSL server;
                    0x03  both SSL client and SSL server

Return:
    buffer size
```

4.6. espconn_secure_connect

```
Function:
    Secure connect (SSL) to a TCP server (ESP8266 is acting as TCP
    client.)

Note:
    • Only one connection is allowed when ESP8266 as SSL client, please
      call espconn_secure_disconnect first, if you want to create
      another SSL connection.
```

- If SSL encrypted packet size is larger than ESP8266 SSL buffer size (default 2KB, set by espconn_secure_set_size), SSL connection will fail, will enter espconn_reconnect_callback

Prototype:

```
sint8 espconn_secure_connect (struct espconn *espconn)
```

Parameters:

struct espconn *espconn : corresponding connected control block structure

Return:

0 : succeed

Non-0 : error code

ESPCONN_MEM – Out of memory

ESPCONN_ISCONN – Already connected

ESPCONN_ARG – illegal argument, can't find TCP connection according to structure espconn

4.7. espconn_secure_send

Function: send encrypted data (SSL)

Note:

Please call espconn_secure_send after espconn_sent_callback of the pre-packet.

Prototype:

```
sint8 espconn_secure_send (
        struct espconn *espconn,
        uint8 *psent,
        uint16 length
)
```

Parameters:

struct espconn *espconn : corresponding connected control block structure

uint8 *psent : sent data pointer

uint16 length : sent data length

Return:

0 : succeed

Non-0 : error code ESPCONN_ARG – illegal argument, can't find TCP connection according to structure espconn

4.8. espconn_secure_disconnect

Function: secure TCP disconnection(SSL)

Prototype:

```
sint8 espconn_secure_disconnect(struct espconn *espconn)
```

Parameters:

struct espconn *espconn : corresponding connected control block structure

Return:

0 : succeed

Non-0 : error code ESPCONN_ARG – illegal argument, can't find TCP connection according to structure espconn