REPORT Cyber Security Activity Project
Andrea Zanni - a.a. 2022/2023

# Creazione dei certificati e loro gestione

Ho seguito queste guide per 1) creare una CA e un 2) certificato auto-firmato per sslserver.home:

1) https://node-security.com/posts/openssl-creating-a-ca/
2) https://node-security.com/posts/openssl-creating-a-host-certificate/

```
basicConstraints = CA:TRUE
keyUsage = cRLSign, keyCertSign
[req]
distinguished_name = req_distinguished_name
prompt = no
[req_distinguished_name]
C   = IT
ST  = Marche
L   = Gabicce Mare
CN  = My personal CA for IoT
```

```
$ openssl x509 -req -sha512 -days 50 -in host.csr -CA ../../main/ca.crt -
CAkey ../../main/ca.key -CAcreateserial -out host.crt -extfile host-ext.conf
Certificate request self-signature ok
subject=C = IT, ST = Marche, L = Gabicce Mare, O = My Company, OU = My Division,
CN = sslserver.home
```

Siccome, come vedremo più avanti, la board utilizza lo stesso certificato della CA utilizzata per erogare il certificato di sslserver.home, la nostra board riuscirà ad autenticare sslserver.home.
In questo caso la CA siamo noi e dobbiamo mettere in sicurezza la chiave ca.key. Un modo per metterla in sicurezza potrebbe essere quello di crittarla tramite il comando openssl, utilizzato fino ad ora per creare I certificati. Lo stesso vale per la chiave dell'host: host.key. Un altro modo potrebbe essere quello di mettere le chiavi in una specifica cartella con gli opportuni permessi settati.

# Setting up

Lavorerò sulla board: az-delivery-devkit-v4
Con platformio andava specificato il baud rate altrimenti si vedevano caratteri e caratteri non stampabili su terminale. Ciò sembra scaturire dal fatto che la board e l'IDE andassero a due baud rate differenti ma mi fa strano in quanto il baud rate standard di monitoring di solito vale 115200.
Platformio.ini
monitor_speed = 115200

Per buildare, flashare la build sulla board e per monitorare l'output della board occorrono I seguenti comandi:

```
$ idf.py build
$ idf.py flash
$ idf.py monitor
```

La idf.py build utilizza a sua volta una serie di tool per costruire il file .bin finale. Segue un estratto del penultimo e dell'ultimo passo di building, evidenzio le parti interessanti:

```
[841/842] Generating binary image from built executableesptool.py v4.4
Creating esp32 image...
Merged 25 ELF sections
Successfully created esp32 image.
Generated /home/chloe/Desktop/Università/Cyber_Security_M/Attivita_Progettuale/
esp_tls/esp-tls-test/build/esp-tls-test.bin
[842/842] cd /home/chloe/Desktop/Universit...sp_tls/esp-tls-test/build/esp-tls-
test.binesp-tls-test.bin binary size 0xc4210 bytes. Smallest app partition is
0x100000 bytes. 0x3bdf0 bytes (23%) free.
```

La idf.py flash e monitor ricercano da sole la porta su cui si trova la board, altrimenti specificabile con e.g. -p /*dev*/ttyUSB0
Flashare la board con un terminale di monitor attivo produce il seguente errore di esptool, il tool che implementa il protocollo per flashare il firmware sulla board. Occorre quindi chiudere il terminale di monitor e poi flashare il firmware sulla board.

```
CMake Error at run_serial_tool.cmake:55 (message):

 /home/chloe/.espressif/python_env/idf5.0_py3.10_env/bin/python;;/home/chloe/
esp/esp-idf/components/esptool_py/esptool/esptool.py;--chip;esp32
failed
```

Un modo piu intelligente consiste nel:
`$ idf.py flash && idf.py monitor`
o come lo chiama PlatformIO “Upload & Monitor”.

# Embed di dati binari su ESP32

https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-guides/build-system.html#embedding-binary-data

Embedding binary data con PlatformIO non ha funzionato pur avendo seguito la loro documentazione. La build con PlatformIO è la seguente:
Equivalente di click col tasto sinistro del mouse su ‘Build’ nella interfaccia grafica dell’estensione di PlatformIO su VSCode:

```
platformio run --environment az-delivery-devkit-v4
 *  Executing task: platformio run --environment az-delivery-devkit-v4

Processing az-delivery-devkit-v4 (platform:
https://github.com/platformio/platform-espressif32.git; board: az-delivery-
devkit-v4; framework: espidf)
------------------------------------------------------------
Verbose mode can be enabled via `-v, --verbose` option
[...]
Compiling .pio/build/az-delivery-devkit-v4/src/main.o
Linking .pio/build/az-delivery-devkit-v4/firmware.elf
/home/chloe/.platformio/packages/toolchain-xtensa-esp32/bin/../lib/gcc/xtensa-
esp32-elf/11.2.0/../../../../xtensa-esp32-elf/bin/ld: .pio/build/az-delivery-
devkit-v4/src/main.o:(.literal.app_main+0x28): undefined reference to
`_binary_ca_crt_end'
/home/chloe/.platformio/packages/toolchain-xtensa-esp32/bin/../lib/gcc/xtensa-
esp32-elf/11.2.0/../../../../xtensa-esp32-elf/bin/ld: .pio/build/az-delivery-
```

```
devkit-v4/src/main.o:(.literal.app_main+0x2c): undefined reference to
`_binary_ca_crt_start'
collect2: error: ld returned 1 exit status
*** [.pio/build/az-delivery-devkit-v4/firmware.elf] Error 1
```

La build con idf.py mi viene detto che non riesce a trovare il main:

```
/home/chloe/.espressif/tools/xtensa-esp32-elf/esp-2022r1-11.2.0/xtensa-esp32-
elf/bin/../lib/gcc/xtensa-esp32-elf/11.2.0/../../../../xtensa-esp32-elf/bin/ld:
esp-idf/freertos/libfreertos.a(port_common.c.obj): in function `main_task':
/home/chloe/esp/esp-idf/components/freertos/FreeRTOS-Kernel/portable/
port_common.c:128: undefined reference to `app_main'
collect2: error: ld returned 1 exit status
```

La build in un nuovo progetto inizializzato con idf.py ha funzionato:

```
[WIFI] Connecting to WiFi...
I (594) wifi:wifi driver task: 3ffc00e8, prio:23, stack:6656, core=0
I (594) system_api: Base MAC address is not set
[...]
I (5804) wifi:state: auth -> assoc (0)
I (5814) wifi:state: assoc -> run (10)
I (5934) wifi:connected with Wind3 HUB-D44CE1, aid = 19, channel 10, BW20, bssid
= 10:71:b3:d4:4c:e1
I (5934) wifi:security: WPA3-SAE, phy: bgn, rssi: -74
I (6024) wifi:pm start, type: 1

I (6024) wifi:AP's beacon interval = 102400 us, DTIM period = 1
I (6584) esp_netif_handlers: sta ip: 192.168.1.149, mask: 255.255.255.0, gw:
192.168.1.1
[WIFI] Connected to WiFi
[TLS] initialised successfully
[TLS] set global CA store successful
```

Si riporta il CmakeLists.txt rispettivamente della cartella madre e della cartella main

```
cmake_minimum_required(VERSION 3.16)
include($ENV{IDF_PATH}/tools/cmake/project.cmake)
project(esp-tls-test)
target_add_binary_data(esp-tls-test.elf "main/ca.crt" TEXT)
```

---

```
idf_component_register(SRCS "esp-tls-test.c"
INCLUDE_DIRS ".")
```

Nota:
In seguito alla verifica del corretto funzionamento sono state rimosse le printf soprastanti relative a TLS.

---

Dal file ca.crt.S:

```
.section .rodata.embedded
```

il file ca.crt viene incastonato nella sezione .rodata.embedded della memoria flash del firmware proprio come accade con il x509_crt_bundle.S. Non sapendo come verificare il nome per intero, nella parte sottostante ho evidenziato le sezioni più plausibili per il contenimento del certificato della CA all'interno del firmware:

```
$ readelf -S esp-tls-test.elf
There are 111 section headers, starting at offset 0x76a1b4:

Section Headers:
  [Nr] Name              Type            Addr     Off    Size   ES Flg Lk Inf Al
  [ 0]                   NULL            00000000 000000 000000 00      0   0  0
  [ 1] .rtc.text         PROGBITS        400c0000 0c6010 000000 00   W  0   0  1
  [ 2] .rtc.dummy        PROGBITS        3ff80000 0c6010 000000 00   W  0   0  1
  [ 3] .rtc.force_fast   PROGBITS        3ff80000 0c6010 000000 00   W  0   0  1
  [ 4] .rtc.data         PROGBITS        50000000 0c6000 000010 00  WA  0   0  8
  [ 5] .rtc_noinit       PROGBITS        50000010 0c6010 000000 00   W  0   0  1
  [ 6] .rtc.force_slow   PROGBITS        50000010 0c6010 000000 00   W  0   0  1
  [ 7] .iram0.vectors    PROGBITS        40080000 026000 000403 00  AX  0   0  4
  [ 8] .iram0.text       PROGBITS        40080404 026404 014997 00  AX  0   0  4
  [ 9] .dram0.data       PROGBITS        3ffb0000 022000 003b48 00  WA  0   0 16
  [10] .ext_ram_noinit   PROGBITS        3f800000 0c6010 000000 00   W  0   0  1
  [11] .noinit           PROGBITS        3ffb3b48 0c6010 000000 00   W  0   0  1
  [12] .ext_ram.bss      PROGBITS        3f800000 0c6010 000000 00   W  0   0  1
  [13] .dram0.bss        NOBITS          3ffb3b48 025b48 003f98 00  WA  0   0  8
  [14] .flash.appdesc    PROGBITS        3f400020 001020 000100 00   A  0   0 16
  [15] .flash.rodata     PROGBITS        3f400120 001120 020de0 00  WA  0   0 16
  [16] .flash.rodat[...] NOBITS          3f420f00 021f00 001df2 00   A  0   0  1
  [17] .flash.text       PROGBITS        400d0020 03b020 08978b 00  AX  0   0  4
  [18] .phyiram.20       PROGBITS        401597ac 0c47ac 000061 00  AX  0   0  4
  [19] .phyiram.18       PROGBITS        40159810 0c4810 00010e 00  AX  0   0  4
  [20] .phyiram.19       PROGBITS        40159920 0c4920 000090 00  AX  0   0  4
(continua …)
```

Total size ottenuta con idf.py size, nel codice mancava ancora la parte di impostazione del server DNS e la comunicazione con il server. Ho evidenziato la sezione contenente il certificato della CA.

```
Total sizes:
Used static DRAM:   31472 bytes ( 149264 remain, 17.4% used)
      .data size:   15192 bytes
      .bss  size:   16280 bytes
Used static IRAM:   85402 bytes (  45670 remain, 65.2% used)
      .text size:   84375 bytes
   .vectors size:    1027 bytes
Used Flash size :  704893 bytes
      .text     :  562315 bytes
      .rodata   :  142322 bytes
Total image size:  805487 bytes (.bin may be padded larger)
```

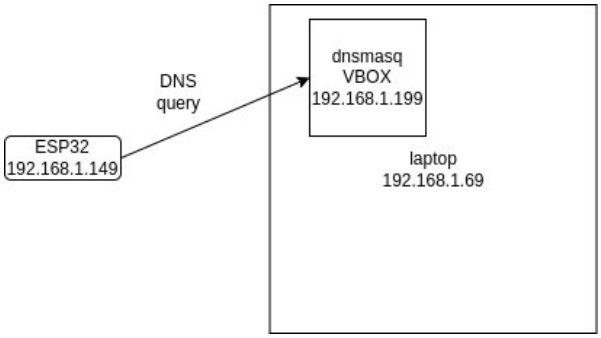
# DNS

Il seguente link riporta I dominii utilizzabili per I dispositivi IoT presenti nelle reti delle nostre case senza farli collidere con mDNS:

https://www.rfc-editor.org/rfc/rfc6762#appendix-G

Ho seguito la guida https://stevessmarthomeguide.com/home-network-dns-dnsmasq/

Nella figura sottostante riporto la semplice architettura utilizzata per far risolvere il nome sslserver.home al server DNS su macchina virtuale debian 11 su virtualbox. Porta DNS: 53. Per fare ciò ho dovuto cambiare il server DNS di default sulla board ESP32 da 192.168.1.1 (il gateway) a 192.168.1.199 (la macchina virtuale).




Nella figura sottostante riporto il traffico di rete, catturato con Wireshark, generato dalla board per ottenere l'indirizzo IP di sslserver.home.

eth.src == ec:62:60:84:dd:ac || eth.dst == ec:62:60:84:dd:ac

| No. | Time | Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|---|---|
| 833 | 9.843657721 | Espressi_84:dd:ac | Broadcast | ARP | 42 | Who has 192.168.1.149? (ARP Probe) |
| 851 | 10.245303749 | Espressi_84:dd:ac | Broadcast | ARP | 42 | Who has 192.168.1.149? (ARP Probe) |
| 857 | 10.662847567 | Espressi_84:dd:ac | Broadcast | ARP | 42 | Who has 192.168.1.149? (ARP Probe) |
| 861 | 10.753011185 | Espressi_84:dd:ac | Broadcast | ARP | 42 | ARP Announcement for 192.168.1.149 |
| 948 | 10.860381295 | Espressi_84:dd:ac | Broadcast | ARP | 42 | Who has 192.168.1.199? Tell 192.168.1.149 |
| 970 | 10.861070786 | IntelCor_8f:b5:7e | Espressi_84:dd:ac | ARP | 60 | 192.168.1.199 is at ac:74:b1:8f:b5:7e |
| 1004 | 10.962172248 | 192.168.1.149 | 192.168.1.199 | DNS | 74 | Standard query 0xa5fb A sslserver.home |
| 1006 | 10.963030915 | 192.168.1.199 | 192.168.1.149 | DNS | 90 | Standard query response 0xa5fb A sslserver.home A 192.168.1.69 |
| 1007 | 10.963102921 | 192.168.1.199 | 192.168.1.149 | DNS | 90 | Standard query response 0xa5fb A sslserver.home A 192.168.1.69 |
| 1008 | 10.966989028 | 192.168.1.149 | 192.168.1.199 | ICMP | 70 | Destination unreachable (Port unreachable) |
| 1010 | 11.064428401 | Espressi_84:dd:ac | Broadcast | ARP | 42 | Who has 192.168.1.69? Tell 192.168.1.149 |
| 1011 | 11.064469778 | IntelCor_8f:b5:7e | Espressi_84:dd:ac | ARP | 42 | 192.168.1.69 is at ac:74:b1:8f:b5:7e |
| 1013 | 11.167637817 | 192.168.1.149 | 192.168.1.69 | TCP | 58 | 63722 → 6000 [SYN] Seq=0 Win=5744 Len=0 MSS=1440 |
| 1014 | 11.167727816 | 192.168.1.69 | 192.168.1.149 | TCP | 54 | 6000 → 63722 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0 |
| 1019 | 11.676014655 | Espressi_84:dd:ac | Broadcast | ARP | 42 | ARP Announcement for 192.168.1.149 |
| 1188 | 13.726883826 | Espressi_84:dd:ac | Broadcast | ARP | 42 | ARP Announcement for 192.168.1.149 |
| 1567 | 15.971920441 | IntelCor_8f:b5:7e | Espressi_84:dd:ac | ARP | 60 | Who has 192.168.1.149? Tell 192.168.1.199 |
| 1578 | 16.079953738 | Espressi_84:dd:ac | IntelCor_8f:b5:7e | ARP | 42 | 192.168.1.149 is at ec:62:60:84:dd:ac |
| 7746 | 63.899643801 | Espressi_84:dd:ac | Broadcast | ARP | 42 | ARP Announcement for 192.168.1.149 |

La cattura è avvenuta quando sul portatile non era stato ancora preparato il server sslserver.home con IP 192.168.1.69 infatti la connessione TCP (pacchetto numero 1014) viene resettata.

# Server

Ho scritto il server. Per validarlo ho utilizzato il seguente comando in locale

```
$ openssl s_client -CAfile certs/ca.crt -connect 0.0.0.0:6000
```

# Put it al'together!

Il seguente è l'output del client ESP32 quando si interrompe l'handshake con il server facendo terminare l'esecuzione del server. Notiamo che dietro le quinte la board ESP32 verifica il certificato del server (evidenziato), certificato associato al dominio sslserver.home.

```
I (6164) wifi:state: auth -> assoc (0)
I (6184) wifi:state: assoc -> run (10)
I (6294) wifi:connected with Wind3 HUB-D44CE1, aid = 19, channel 5, 40U, bssid =
10:71:b3:d4:4c:e1
I (6294) wifi:security: WPA3-SAE, phy: bgn, rssi: -69
I (6384) wifi:pm start, type: 1

I (6384) wifi:AP's beacon interval = 102400 us, DTIM period = 1
```

```
I (7094) esp_netif_handlers: sta ip: 192.168.1.149, mask: 255.255.255.0, gw:
192.168.1.1
[WIFI] Connected to WiFi
W (7414) wifi:<ba-add>idx:0 (ifx:0, 10:71:b3:d4:4c:e1), tid:0, ssn:5, winSize:64
E (23074) esp-tls-mbedtls: mbedtls_ssl_handshake returned -0x0050
I (23084) esp-tls-mbedtls: Certificate verified.
E (23084) esp-tls: Failed to open new connection
E (23084) esp-tls-mbedtls: write error :-0x0050:
bye ;)
```

Se invece non interrompiamo niente allora il server stampa a video il messaggio del cliente e il cliente termina correttamente la sua esecuzione.

Codice sorgente in questa repository https://github.com/andreaz98/esp32-tls

```
bye ;)
```