

```
1
2
3 void PassChain_IoT() {
4
5     /* Presentazione di Alberto Montefusco */
6
7     const char *corso = 'IoT Security';
8
9     const char *Prof = 'Christiancarmine Esposito';
10
11    const char *aa = '2022 - 2023';
12
13    Presentation.begin(course, Prof, aa);
14 }
```



Table Of 'Contents' {

01 Introduzione

/* Introduzione ai Password Manager e a PassChain */

02 Hardware

/* Sensori usati e descrizione del circuito finale */

03 Software

/* Librerie usate, comunicazione SSL, server Python */

04 Vulnerabilità_&_Sicurezza

/* Attacchi e Soluzioni Sicure */

}

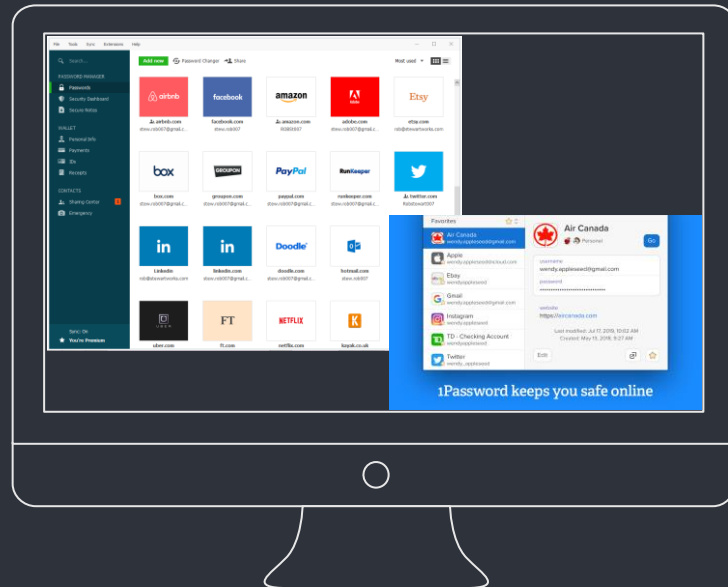
```
1      01 {
2
3
4
5      [Introduzione]
6
7
8      /* Cos'è un PassWord Manager
9         e il dispositivo PassChain */
10
11
12     }
13
14
```

```
1 Password_Manager {
2
3     /* Cosa sono? */
4
5     Serial.println('Servizi online che permettono
6                     all'utente di salvare password');
7
8     delay(100);
9
10
11     Serial.println(' e sincronizzarle su tutti i
12                    dispositivi personali');
13
14 }
```

```
1  
2  
3  
4 tft.pushImage (
```

```
5  
6  
7  
8  
9 Dashlane e 1Password  
10 sono i Password  
11 Manager più famosi
```

```
12  
13  
14 );
```



```
1 Serial.println ( 'Vantaggi: ' +
2
3   Step 01  Le App sono disponibili per tutte
4   le piattaforme
5
6   Step 02  Alcune estensioni Browser permettono di
7   inserire le password automaticamente
8
9   Step 03  Le password sono sincronizzate su
10  tutti i dispositivi
11
12  Step 04  Accesso veloce
13
14 );
```

```
1 Serial.println ( 'Svantaggi: ' +
```

```
2  
3 — Step 01 I servizi online e cloud possono  
4           essere violati anche da remoto
```

```
5  
6 — Step 02 Installare un'applicazione o un'estensione  
7           su ciascuno dei propri dispositivi personali
```

```
8  
9 — Step 03 Meno pratico quando si usano  
10          dispositivi non personali
```

```
11  
12 — Step 04 Sigle Point of Failure perché l'accesso a  
13          tutte le tue password è protetto da  
14          un'unica password sicura
```

```
);
```

PassChain_IoT {

/* Cos'è PassChain? */

Serial.println('PassChain è un dispositivo IoT con l'obiettivo di
facilitare l'utente durante l'autenticazione digitale');

delay(100);

Serial.println('ma anche di garantire la loro sicurezza attraverso la sua
funzione di password manager');



Bluetooth

< Connessione ad altri dispositivi
tramite Bluetooth >



Autenticazione

< Ogni operazione è compiuta tramite
autenticazione con Fingerprint >



Crittografia

< Tutti i dati sono cifrati con lo
schema AES 128 bit – GCM >

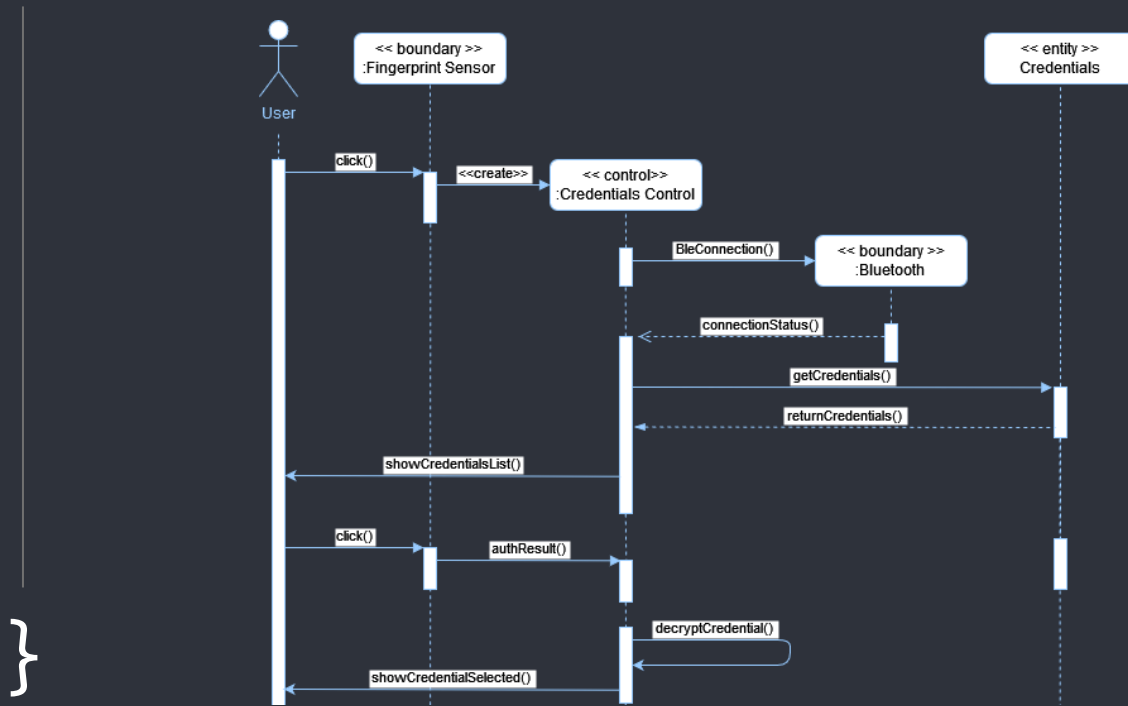


Comunicazione SSL

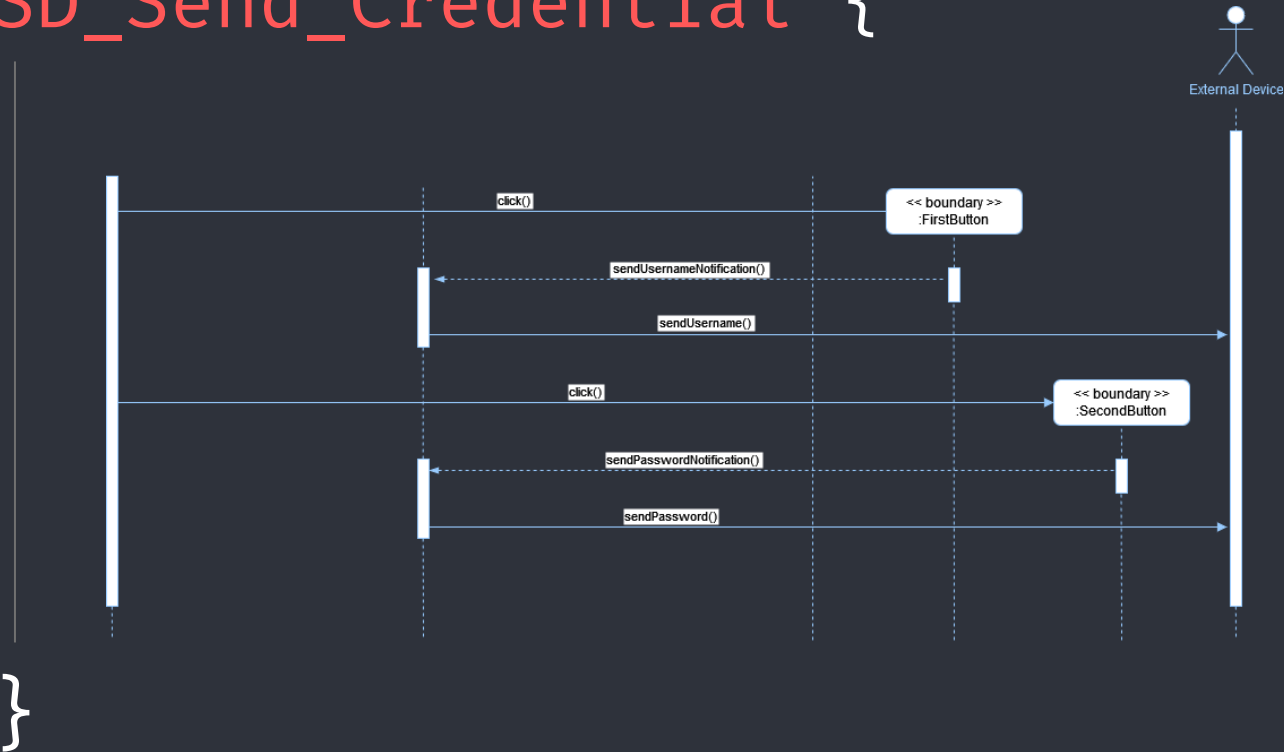
< SSL garantisce una comunicazione
sicura con il server python >

}

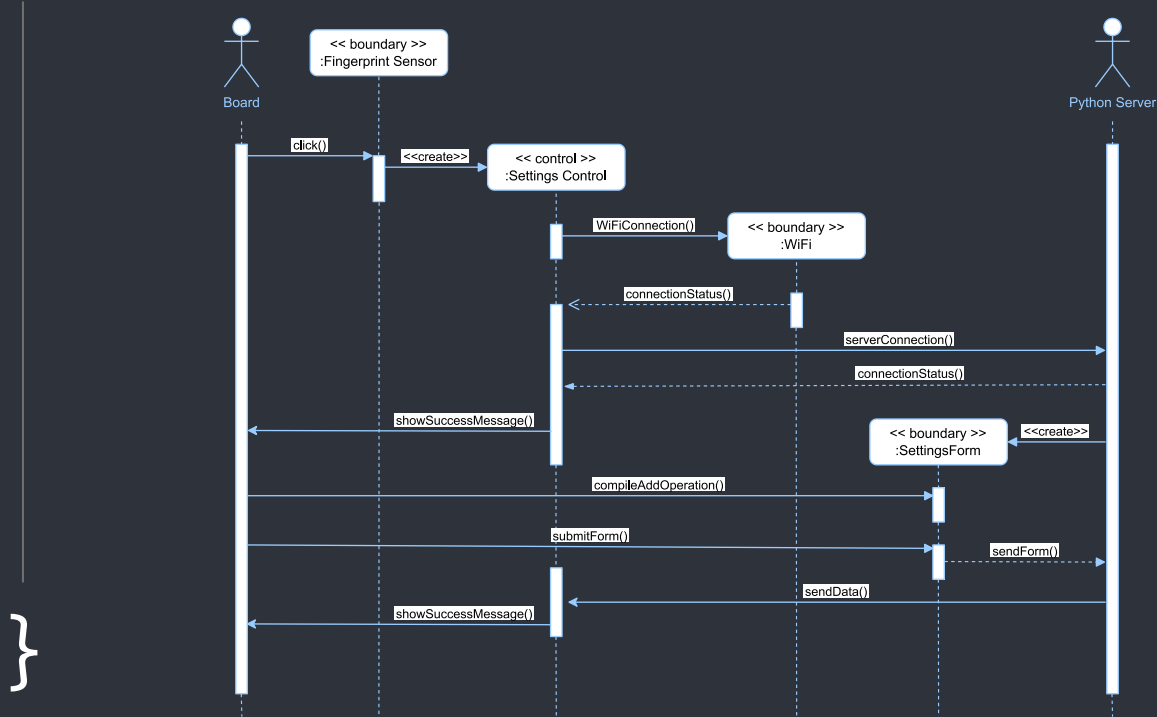
SD_Send_Credential {



SD_Send_Credential {



SD_Server_Operation {



```
1      02 {
2
3
4
5      [Hardware]
6
7
8      /* Lora TTgo Esp32 16 Mb,
9         Fingerprint, Battery */
10
11
12     }
13
14
```

```
1 void Lora_TTgo_Esp32 () {
```



```
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14
```

< La scheda ha un display OLED integrato da 0,96'', due pulsanti programmabili e uno per il riavvio, un connettore di alimentazione per le batterie e una porta di tipo C per il collegamento del dispositivo e l'alimentazione della scheda stessa. Ha una memoria di 16 Mb con SPIFFS come file system >

```
15 void Fingerprint () {
```



```
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65  
66  
67  
68  
69  
70  
71  
72  
73  
74  
75  
76  
77  
78  
79  
80  
81  
82  
83  
84  
85  
86  
87  
88  
89  
90  
91  
92  
93  
94  
95  
96  
97  
98  
99  
100
```

< Il sensore di impronte digitali consente di autenticare l'utente garantendone l'unicità. Il sensore può memorizzare 126 impronte digitali: una di queste serve per tornare indietro nelle impostazioni >

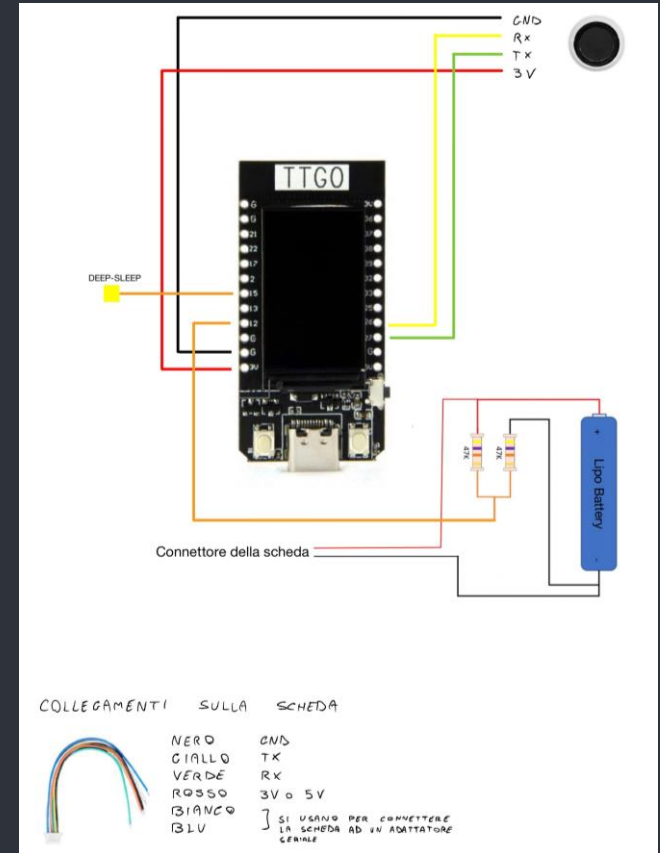
```
1 void Lora_Ttgo_Esp32 () {
2
3    < La batteria LiPo da 500 mAh e 3,7 V consente di
4   avere un dispositivo portatile e indipendente >
5
6 }
7
8 void show_circuit (Board esp32, Sensor fingerprint,
9                   Sensor battery) {
10
11   Circuit circuit = new Circuit (Esp32, fingerprint, battery);
12   tft.pushImage(circuit.show());
13
14 }
```

```

1
2
3
4 tft.pushImage (
5
6
7
8
9
10
11
12
13
14 );

```

- GPIO 15: placca metallica per riattivare la board quando entra in modalità deep_sleep
- GPIO 12: circuito della batteria

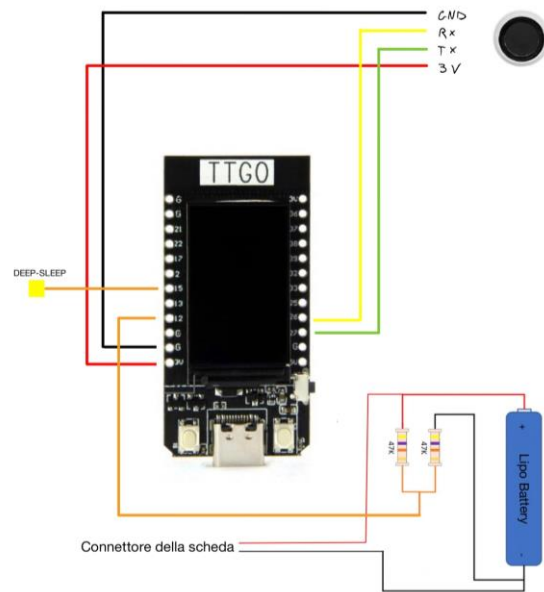


```

1
2
3
4 tft.pushImage (
5
6
7
8
9
10
11
12
13
14 );

```

- G: connessione con GND
- La line rossa è collegata alla 3V
- GPIO 26: TX della scheda
- GPIO 27: RX della scheda



COLLEGAMENTI SULLA SCHEDA



NERO	GND
GIALLO	TX
VERDE	RX
ROSSO	3V o 5V
BIANCO	} SI USANO PER COLLEGARE LA SCHEDA AD UN ADATTATORE CEE-ME
BLU	


```
1      03 {
2
3
4
5      [Software]
6
7
8      /* Librerie usate, comunicazione SSL,
9       server Python */
10
11
12     }
13
14
```

```
1 Libraries_used(bool external) {
```

```
2  
3     if (!external) { /* dettagli librerie Arduino */
```

```
4  
5         < TFT_eSPI.h >      * Libreria Arduino usata per scrivere  
6                             * testo o visualizzare immagini sul  
7                             * display OLED.
```

```
8  
9         < SD.h >           * Libreria Arduino per leggere e scrivere  
10                            * in memoria. Il file system usato è  
11                            * SPIFFS.
```

```
12        < ArduinoJson.h >  * Libreria Arduino usata per gestire i  
13                            * file JSON, nei quali sono memorizzate  
14                            * le credenziali utente e dell'Hotspot
```

```
1
2  < WiFiClientSecure.h > * Libreria Arduino usata per creare e
3                          * fare il setup della comunicazione SSL
4
5      < mbedtls/gcm.h > * Cifratura e decifratura dei dati con lo
6                          * schema AES - 128 bit GCM mode
7
8  } else { /* dettagli delle librerie esterne */
9
10      < BleKeyboard.h > * Simulare una tastiera virtuale e
11                        * inviare i messaggi tramite Bluetooth
12
13      < Adafruit_Fingerprint.h > * Gestione delle impronte digitali
14  }
```

SSL_Communication {

```
1  
2  
3  /*****  
4  * Ho creato una comunicazione sicura tra l'Esp32 e il dispositivo *  
5  * esterno che funge da server per fare il setup dei dati. *  
6  *****/
```

Creazione Certificati



< Certificati CA, client
e chiave associata del
client >

Salvataggio Certificati



< I certificati sono
salvati nel codice Arduino
e inviati al server >

Start Server Python



< Caricamento del certificato
del server e verifica di esso
e del certificato client >

Recezione dati



< Il server crea una
socket e riceve i
dati dall'ESP32 >

```
14 }
```

Server_Python {

```
1  /*****  
2  
3  * Permette di effettuare il setup della scheda. Le operazioni *  
4  * riguardano: aggiunta, modifica ed eliminazione delle credenziali, *  
5  * setup del fingerprint, setup credenziali dell'hotspot *  
6  *****/
```

Start server



< Stringa di benvenuto,
numero di fingerprint, JSON
con le credenziali >

Operazioni CRUD



< Per il setup Esp32, possiamo
eseguire operazioni CRUD sui
dati >

Setup Fingerprints



< Il Server verifica se ci
sono almeno 2 FP:
- conferma delle operazioni
- tasto per tornare indietro >

Setup Hotspot



< Il Server può modificare
l'SSID e la password
dell'hotspot >

}

1 04 {
2
3

4
5 [Vulnerabilità &
6 Sicurezza]
7
8

9 /* Attacchi e
10 Soluzioni Sicure */
11

12 }
13
14

```
1 void MITM_attack (Adv, Alice, Bob) {
```

```
2  
3  /*******  
4   * PassChain è vulnerabile all'attacco MITM su Bluetooth perché      *  
5   * manca l'autenticazione tra i due dispositivi. Una soluzione è      *  
6   * utilizzare BLE v4.2 BR/EDR utilizzando la crittografia basata su    *  
7   * ECDH e utilizzando il modello di binding di confronto numerico     *  
8   * per l'autenticazione                                                *  
9   *****/
```

```
10  
11     < Questa soluzione non è forte perchè la v4.2 fino alla 5.0 soffrono di  
12     una vulnerabilità chiamata "BLURtooth" che sfrutta il CTDK >
```

```
13  
14     < Questo debolezza è stata risolta dal  
15     Protocollo GATT, disponibile nelle ultime  
16     versioni dei dispositivi dotati di BLE >
```

```
17 }
```



```
1 void Exploit_FP (memory) {
2
3     /*****
4     * PassChain usa il Fingerprint per autenticare l'utente quando *
5     * esegue un'operazione. *
6     *****/
7
8     < La vulnerabilità consiste nello sfruttare la comunicazione UART
9     perchè il sensore usa il Seriale TTL per comunicare. Ad esempio,
10    possiamo usare Attify Badge per emulare una connessione seriale ed
11    accedere al dispositivo target >
12
13    < Una soluzione è di isolare PassChain da questi
14    dispositivi, in particolare, l'utente non deve lasciare
15    incustodita la board >
16 }
```



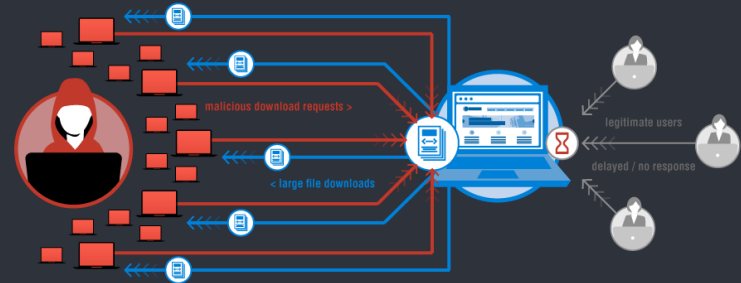

```
1 void DoS_attack (resources) {
```

```
2  
3 /*****  
4  * L'utente può eseguire il setup di PassChain connettendo il      *  
5  * dispositivo al server Python.                                   *  
6  *****/
```

```
7  
8 < Un malintenzionato può eseguire un attacco DoS esauendo le  
9 risorse dell'host che sta eseguendo il server e interrompendo i  
10 servizi dell'host connesso alla rete >
```

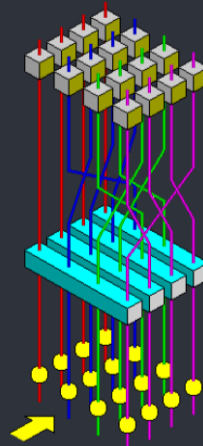
```
11  
12 < In questo modo l'utente non  
13 può connettersi al server ed  
14 eseguire il setup della scheda >
```

```
}
```



```
1 void Security_Solution (crypto, SSL) {
2
3     < Le soluzioni sicure adottate sono l'uso della crittografia
4     e della comunicazione SSL >
5
6     Tag AES_128 (plainText, * I dati sono salvati sul dispositivo
7                           key, IV); * cifrati usando lo schema AES-128 bit con
8                                   * la modalità GCM
9
10
11     void SSL (certs); * I dati sono inviati dal server Python
12                    * all'Esp32 in chiaro perchè la
13                    * comunicazione è realizzata con SSL
14                    * garantendo autenticazione, integrità e
15                    * riservatezza
16 }
```

```
1 Tag AES_128 (plainText, key, IV) {
2
3
4     /*****
5      * AES-GCM è un cifrario a blocchi che garantisce autenticazione,
6      * confidenzialità e integrità.
7      *****/
8
9     < Abbiamo 4 input:
10
11     - Secret key (128-bit);
12
13     - IV (96-bit);
14
15     - Plaintext;
16
17     - Optional addition authenticated data (AAD) >
18 }
```



```
1 Tag AES_128 (plainText, key, IV) {
2
3     /*****
4      * La chiave deve essere generate casualmente, ad esempio utilizzando *
5      * le response PUF in Esp32.                                           *
6      *****/
7
8     < Ciò è utile perché un PUF viene utilizzato per creare chiavi che
9     vengono generate su richiesta e cancellate una volta utilizzate. La
10    chiave PUF non esiste in forma digitale all'interno del circuito
11    integrato ed è derivata e prodotta su richiesta sfruttando le
12    caratteristiche intrinseche degli elementi del circuito >
13
14    < Nel nostro caso abbiamo una chiave di test
15    salvata hard-coded nel codice Arduino >
16 }
```



```
1 Tag AES_128 (plainText, key, IV) {
2
3     /*****
4      * Gli IV sono generati randomicamente sfruttando RGN Hardware *
5      * dell'Esp32. *
6      *****/
7     < La funzione esp_random() produce dei TRN solo se il Wi-Fi o il
8     Bluetooth sono attivi. La funzione ritorna un intero di 32 bit,
9     in questo modo la stringa IV è costruita in append generando
10    altri TRN per produrre in output una stringa di 96 bit >
11
12    < Ogni credenziale ha il suo IV salvato in chiaro
13    e l'IV associato cambia ad ogni cifratura della
14    stessa credenziale >
15 }
```

IV

```
1 Serial.println ( 'Vantaggi AES: ' +
2
3   — Step 01 Algoritmo veloce
4
5
6   — Step 02 La chiave può essere lunga 128 – 192 – 256
7     bit non permettendo attacchi di brute force
8
9   — Step 03 La taglia di ogni blocco è di 128 bit
10
11   — Step 04 La modalità GCM garantisce
12     autenticazione, riservatezza e
13     integrità
14 );
```

```
1 Serial.println ( 'Svantaggi AES: ' +
2
3   Step 01 Se il nonce viene riutilizzato le proprietà
4   di riservatezza e integrità saranno violate
5
6   Step 02 Tag corti producono messaggi fasulli
7
8
9   Step 03 Le implementazioni GCM sono
10  vulnerabili ai timing attacks
11
12  Step 04 GCM è vulnerabile ai cycling attacks
13
14 );
```

```
1 void SSL (certs) {
2
3     /*****
4     * SSL è stato utilizzato per creare un canale sicuro durante lo scambio *
5     * di dati tra il server Python e la scheda.                               *
6     *****/
7
8     < SSL ci permette di autenticare PassChain utilizzando dei
9     certificati. Prima che avvenga lo scambio di dati, il server
10    controlla i certificati client e server >
11
12    < SSL è un protocollo pesante a causa della continua
13    cifratura e decifratura dei dati quando partono e
14    arrivano a destinazione e bisogna generare nuovi
15    certificati per ogni nuova scheda che si connette al
16    server >
17 }
```




```
1 Serial.println ( 'Vantaggi SSL: ' +
2
3
4   Autenticazione  Il client utilizza la chiave pubblica del
5                   server per cifrare i dati; il server utilizza
6                   la chiave pubblica nel certificato client per
7                   decifrare i dati inviati dal client
8
9   Integrità       SSL fornisce l'integrità dei dati calcolando
10                  un digest del messaggio
11
12   Riservatezza   SSL utilizza una combinazione di
13                  crittografia simmetrica e
14                  asimmetrica per garantire la
15                  riservatezza dei messaggi
16 );
```

```
1
2
3 void ringraziamenti () {
4
5
6     Serial.println('Grazie ');
7
8
9     Serial.println('per l'attenzione');
10
11 }
12
13
14
```