PassChain.ino

Details.ino

```
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  }
```

Arduino Language

PassChain.ino

Details.ino

Table Of ‘Contents’ {

}

Arduino Language

```
01 {

   [Introduzione]

      /* Cos'è un PassWord Manager
         e il dispositivo PassChain */

}
```

PassChain.ino

Details.ino

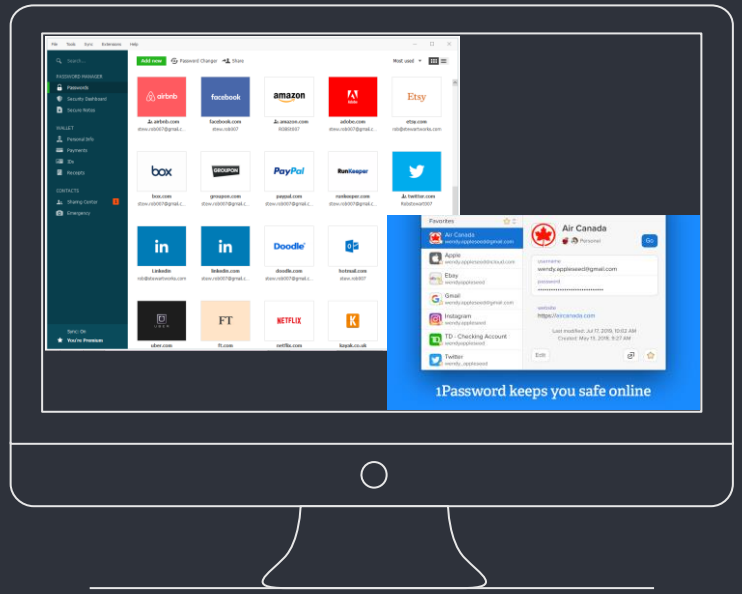
```
Password_Manager {

    /* Cosa sono? */

        Serial.println('Servizi online che permettono
                        all'utente di salvare password');

        delay(100);

         Serial.println(' e sincronizzarle su tutti i
                         dispositivi personali');

}
```

Arduino Language

PassChain.ino

Details.ino

```
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  );
```



Arduino Language

PassChain.ino

Details.ino

```
Serial.println (‘Vantaggi: ‘ +
```

- Step 01 — Le Apps sono disponibili per tutte le piattaforme
- Step 02 — Alcune estensioni Browser permettono di inserire le password automaticamente
- Step 03 — Le password sono sincronizzate su tutti i dispositivi
- Step 04 — Accesso veloce

```
);
```

Arduino Language

PassChain.ino

Details.ino

```
Serial.println (‘Svantaggi: ‘ +
```

Step 01 — I servizi online e cloud possono essere violati anche da remoto

Step 02 — Installare un'applicazione o un'estensione su ciascuno dei propri dispositivi personali

Step 03 — Meno pratico quando si usano dispositivi non personali

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

```
);
```

Arduino Language

PassChain.ino

Details.ino

```
PassChain_IoT {

    /* Cos'è PassChain? */

        Serial.println('PassChain è un dispositivo IoT con l'obbietivo 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

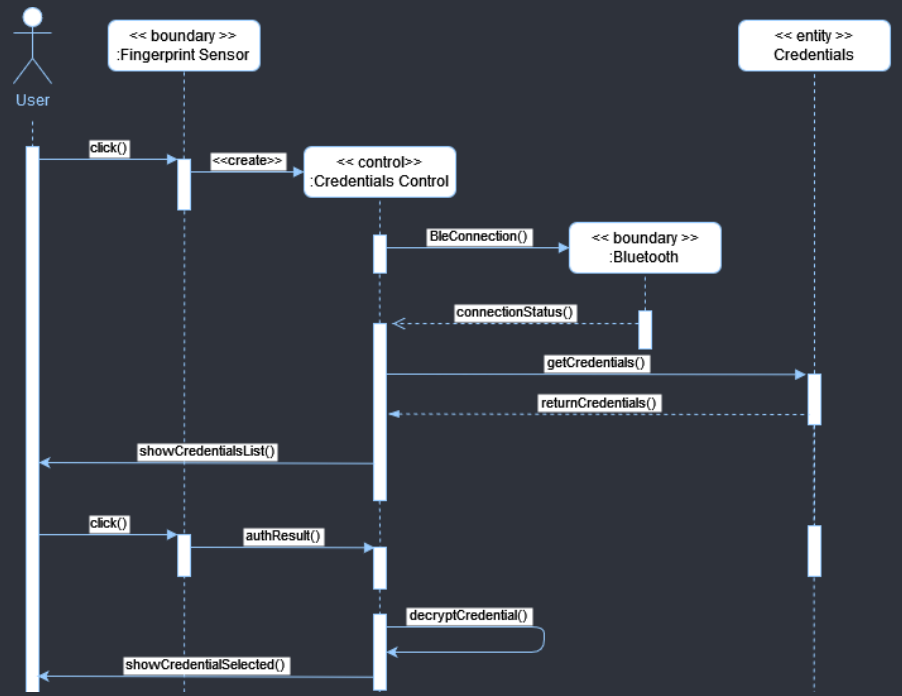
< SSL garantisce una comunicazione sicura con il server python >

Arduino Language

PassChain.ino

Details.ino

SD_Send_Credential {



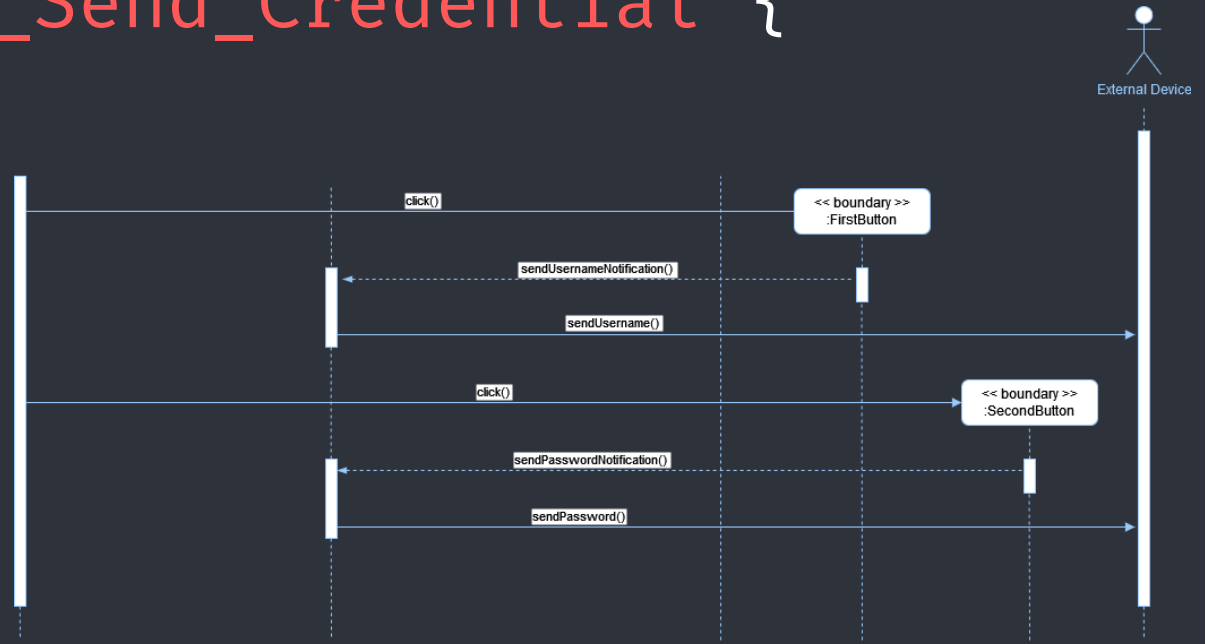
}

Arduino Language

PassChain.ino

Details.ino

SD_Send_Credential {



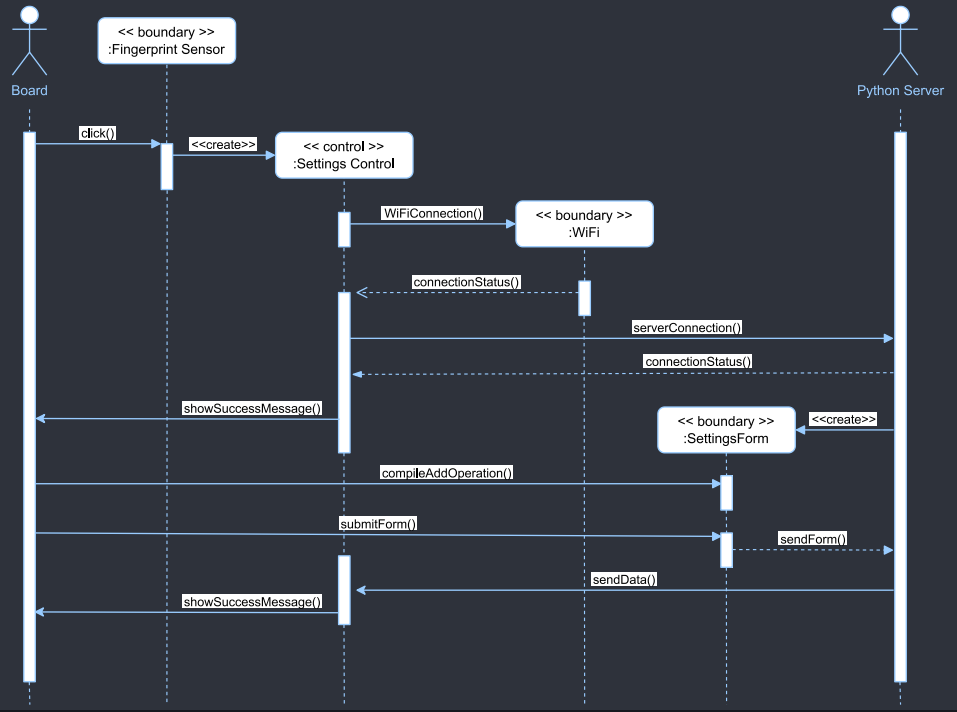
}

Arduino Language

PassChain.ino

Details.ino

SD_Server_Operation {



}

Arduino Language

PassChain.ino

Details.ino

```
02 {

  [Hardware]

    /* Lora TTgo Esp32 16 Mb,
       Fingerprint, Battery */

}
```

Arduino Language

PassChain.ino

Details.ino

```
void Lora_TTgo_Esp32 () {
```

< 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 >

```
}

void Fingerprint () {
```

< 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 >

```
}
```

Arduino Language

PassChain.ino

Details.ino

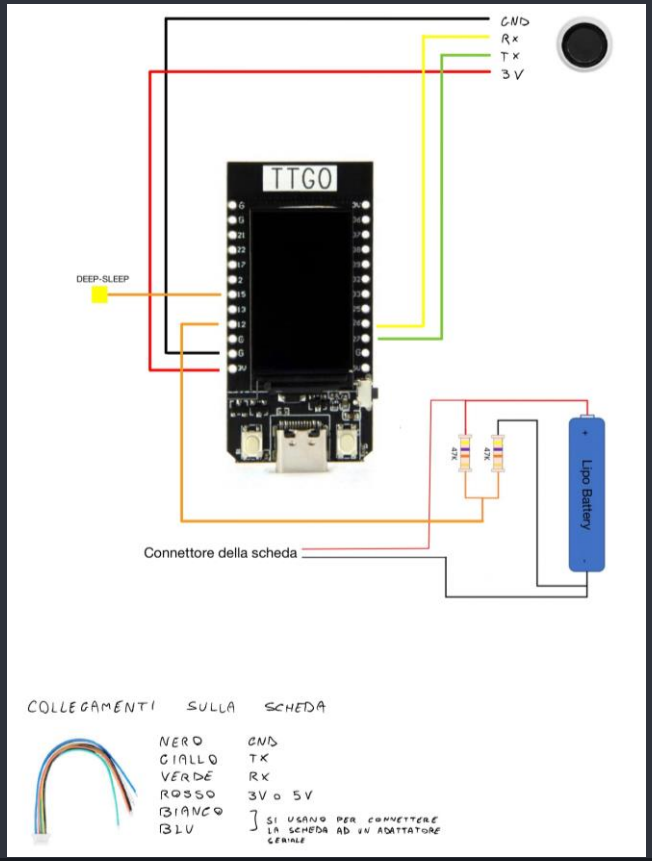
```
void Lora_Ttgo_Esp32 () {

    

    < La batteria LiPo da 500 mAh e 3,7 V consente di
    avere un dispositivo portatile e indipendente >

}
void show_circuit (Board esp32, Sensor fingerprint,
                   Sensor battery) {

    Circuit circuit = new Circuit(Esp32, fingerprint, battery);

    tft.pushImage(circuit.show());

}
```

Arduino Language

PassChain.ino

```
1
2
3
4  tft.pushImage (
5
6
7      • GPIO 15: placca metallica
8        per riattivare la board
9        quando entra in modalità
10       deep_sleep
11
12     • GPIO 12: circuito della
13       batteria
14 );
```

Details.ino



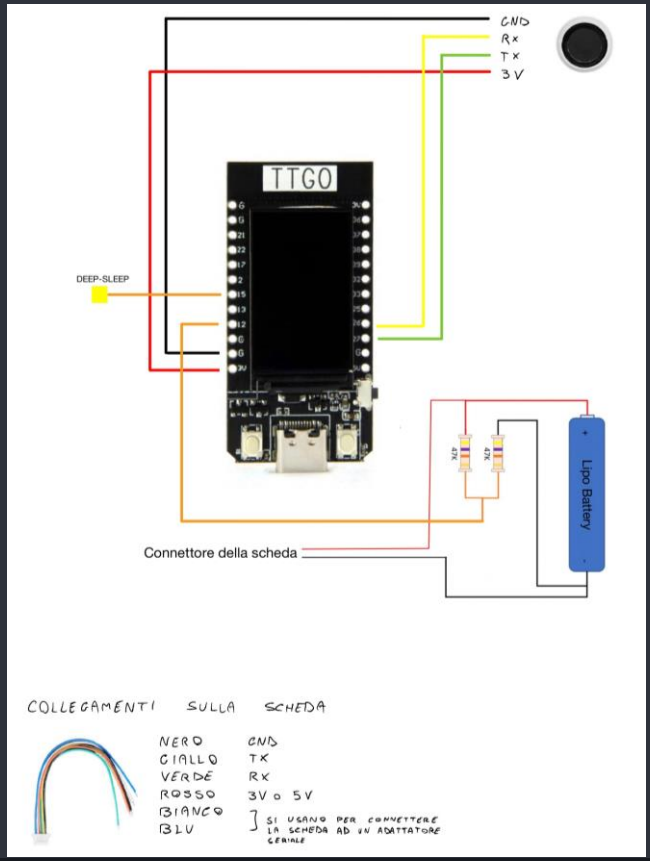
Arduino Language

PassChain.ino

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

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

Details.ino



Arduino Language

```
03 {

    [Software]

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

}
```

PassChain.ino

Details.ino

```
Libraries_used(bool external) {

    if (!external) { /* dettagli librerie Arduino */

                                    * Libreria Arduino usata per scrivere
        < TFT_eSPI.h >              * testo o visualizzare immagini sul
                                    * display OLED.

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

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

Arduino Language

PassChain.ino

Details.ino

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
```

```
< WiFiClientSecure.h >
```

* Libreria Arduino usata per creare e
* fare il setup della comunicazione SSL

```
< mbedtls/gcm.h >
```

* Cifratura e decifratura dei dati con lo
* schema AES - 128 bit GCM mode

```
} else { /* dettagli delle librerie esterne */

< BleKeyboard.h >
```

* Simulare una tastiera virtuale e
* inviare i messaggi tramite Bluetooth

```
< Adafruit_Fingerprint.h >
```

* Gestione delle impronte digitali

```
}
}
```

Arduino Language

PassChain.ino

Details.ino

SSL_Communication {

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

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 >

}

Arduino Language

PassChain.ino

Details.ino

Server_Python {

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

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 >

}

Arduino Language

04 {

[Vulnerabilità & Sicurezza]

/* Attacchi e
Soluzioni Sicure */

}

PassChain.ino

Details.ino

```
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
15     < Questo debolezza è stata risolta dal
16       Protocollo GATT, disponibile nelle ultime
17       versioni dei dispositivi dotati di BLE >
18 }
```

Arduino Language

PassChain.ino

Details.ino

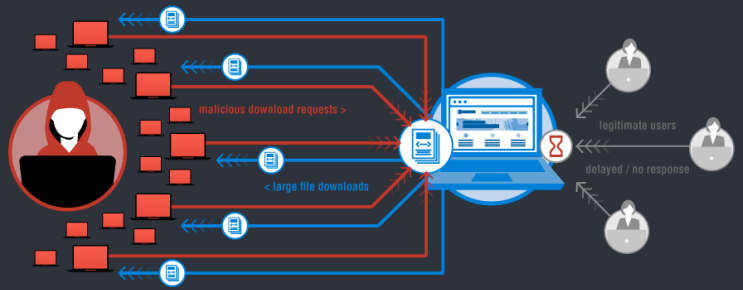
```
void Exploit_FP (memory) {

    /*****************************************************************
     * PassChain usa il Fingerprint per autenticare l'utente quando  *
     * esegue un'operazione.                                         *
     *****************************************************************/

        < La vulnerabilità consiste nello sfruttare la comunicazione UART
        perchè il sensore usa il Seriale TTL per comunicare. Ad esempio,
        possiamo usare Attify Badge per emulare una connessione seriale ed
        accedere al dispositivo target >

          < Una soluzione è di isolare PassChain da questi
            dispositivi, in particolare, l'utente non deve lasciare
            incustodita la board >

}
```

Arduino Language

PassChain.ino

Details.ino

```
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 esaurendo le
9          risorse dell'host che sta eseguendo il server e interrompendo i
10         servizi dell'host connesso alla rete >
11
12
13     < In questo modo l'utente non
14       può connettersi al server ed
         eseguire il setup della scheda >
   }
```



Arduino Language

PassChain.ino

Details.ino

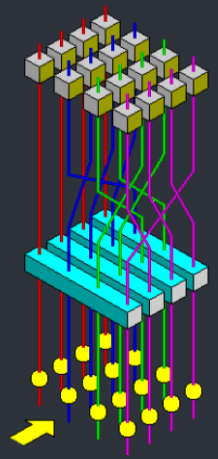
```
void Security_Solution (crypto, SSL) {

    < Le soluzioni sicure adottate sono l'uso della crittografia
    e della cominicazione SSL >


    Tag AES_128 (plainText,
                 key, IV);

                                    * I dati sono salvati sul dispositivo
                                    * cifrati usando lo schema AES-128 bit con
                                    * la modalità GCM


        void SSL (certs);

                                    * I dati sono inviati dal server Python
                                    * all'Esp32 in chiaro perchè la
                                    * comunicazione è realizzata con SSL
                                    * garantendo autenticazione, integrità e
                                    * riservatezza
}
```

Arduino Language

PassChain.ino

Details.ino

```
Tag AES_128 (plainText, key, IV) {

    /*****************************************************************
     * AES-GCM è un cifrario a blocchi che garantisce autenticazione,    *
     * confidenzialità e integrità.                                      *
     *****************************************************************/

        < Abbiamo 4 input:

        - Secret key (128-bit);

        - IV (96-bit);

        - Plaintext;

        - Optional addition authenticated data (AAD) >


}
```



Arduino Language

PassChain.ino

Details.ino

```
Tag AES_128 (plainText, key, IV) {

    /*****************************************************************
     * La chiave deve essere generate casualmente, ad esempio utilizzando *
     * I PUFs in Esp32.                                              *
     *****************************************************************/

        < Ciò è utile perché un PUF viene utilizzato per creare chiavi che
        vengono generate su richiesta e cancellate una volta utilizzate. La
        chiave PUF non esiste in forma digitale all'interno del circuito
        integrato ed è derivata e prodotta su richiesta sfruttando le
        caratteristiche fisiche degli elementi del circuito >

     < Nel nostro caso abbiamo una chiave di test
       salvata hard-coded nel codice Arduino >

}
```

Arduino Language

PassChain.ino | Details.ino

```
Tag AES_128 (plainText, key, IV) {

    /*****************************************************************
     * Gli IV sono generati randomicamente sfruttando RGN Hardware    *
     * dell'Esp32.                                                    *
     *****************************************************************/

        < La funzione esp_random() produce dei TRN solo se il Wi-Fi o il
        Bluetooth sono attivi. La funzione ritorna un intero di 32 bit,
        in questo modo la stringa IV è costruita in append generando
        altri TRN per produrre in output una stringa di 96 bit >


     < Ogni credenziale ha il suo IV salvato in chiaro
       e l'IV associato cambia ad ogni cifratura della
       stessa credenziale >

}
```

IV

Arduino Language

PassChain.ino

Details.ino

```
Serial.println (‘Vantaggi AES: ‘ +
```

Step 01 Algoritmo veloce

Step 02 La chiave può essere lunga 128 – 192 – 296 bit non permettendo attacchi di brute force

Step 03 La taglia di ogni blocco è di 128 bit

Step 04 La modalità GCM garantisce autenticazione, riservatezza e integrità

```
);
```

Arduino Language

PassChain.ino

Details.ino

```
Serial.println (‘Svantaggi AES: ‘ +
```

- Step 01 — Se il nonce viene riutilizzato le proprietà di riservatezza e integrità saranno violate
- Step 02 — Tag corti producono messaggi fasulli
- Step 03 — Le implementazioni GCM sono vulnerabili ai timing attacks
- Step 04 — GCM è vulnerabile ai cycling attacks

```
);
```

Arduino Language

PassChain.ino

Details.ino

```
void SSL (certs) {

    /*****************************************************************
     * SSL è stato utilizzato per creare un canale sicuro durante lo scambio *
     * di dati tra il server Python e la scheda.                     *
     *****************************************************************/

        < SSL ci permette di autenticare PassChain utilizzando dei
        certificate. Prima che avvenga lo scambio di dati, il server
        controlla i certificati client e server >

     < SSL è un protocollo pesante a causa della continua
       cifratura e decifratura dei dati quando partono e
       arrivano a destinazione e bisogna generare nuovi
       certificati per ogni nuova scheda che si connette al
       server >

}
```

Arduino Language

PassChain.ino

Details.ino

```
Serial.println (‘Vantaggi SSL: ‘ +
);
```

- **Autenticazione** — Il client utilizza la chiave pubblica del server per cifrare i dati; il server utilizza la chiave pubblica nel certificato client per decifrare i dati inviati dal client
- **Integrità** — SSL fornisce l'integrità dei dati calcolando un digest del messaggio
- **Riservatezza** — SSL utilizza una combinazione di crittografia simmetrica e asimmetrica per garantire la riservatezza dei messaggi

PassChain.ino

Details.ino

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

Arduino Language