PassChain.ino

Details.ino

```
1
2
3   void PassChain_IoT() {
4
5     /* Presentation of Alberto Montefusco */
6
7     const char *course = '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

Table Of 'Contents' {

}

```
01 {

   [Introduction]

      /* What is a PassWord Manager
         and PassChain device */

}
```

PassChain.ino

Details.ino

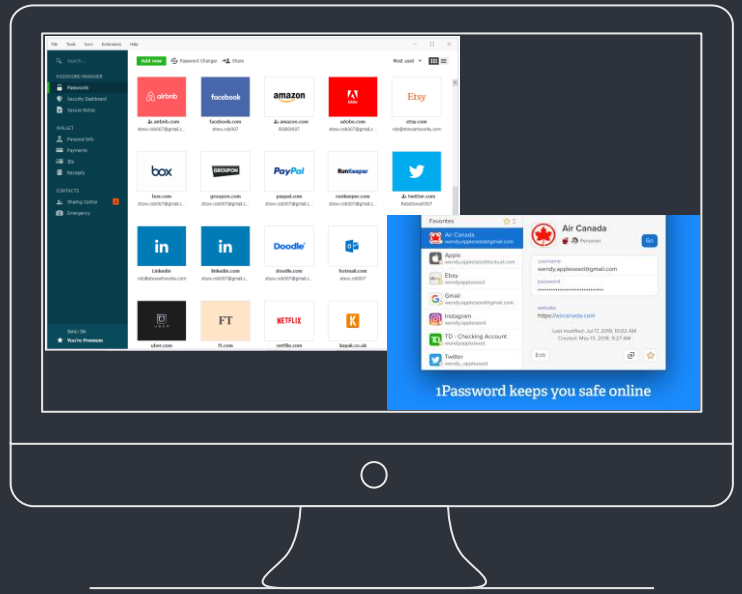
```
Password_Manager {

    /* What are them? */

        Serial.println('Online services that allow
                        users to store password');

        delay(100);

        Serial.println(' and sync them across all your
                         personal devices');

}
```

Arduino Language

PassChain.ino

Details.ino

```
1
2
3
4  tft.pushImage (
5
6
7
8
9      Dashlane and 1Password
10     are most popular
11     password manager
12
13
14 );
```



Arduino Language

PassChain.ino

Details.ino

```
Serial.println (‘Advantages: ‘ +
    Step 01  Apps available for all platforms
        Step 02  Browser extensions allow to
                 automatically enter passwords
            Step 03  Sync on all devices
                Step 04  Fast access
);
```

Arduino Language

PassChain.ino

Details.ino

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

- **Step 01** Online and cloud services can be violated even remotely
- **Step 02** Need to install an application or extension on each of your personal devices
- **Step 03** Less practical when using non-personal devices
- **Step 04** Sigle Point of Failure because access to all of your passwords is protected by a single strong password

```
);
```

Arduino Language

PassChain.ino

Details.ino

PassChain_IoT {

```
/* What is It? */
    Serial.println('PassChain is an IoT device that aims to facilitate
                    the user in digital authentication');

    delay(100);

    Serial.println('but also to ensure their security through
                    its function of password manager');
```

Bluetooth

< Connect to other devices with Bluetooth to send user's credentials >

Authentication

< Every operation is done through Fingerprint authentication >

Cryptography

< All data are encrypted with AES 128 bit – GCM mode >

SSL Communication

< SSL allows secure communication with the python server >

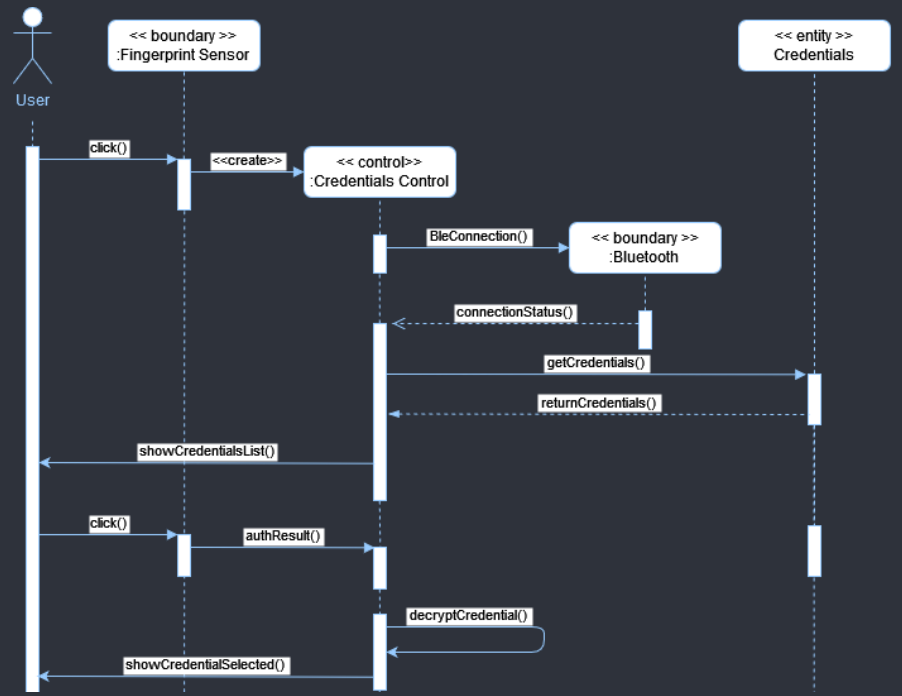
}

Arduino Language

PassChain.ino

Details.ino

SD_Send_Credential {



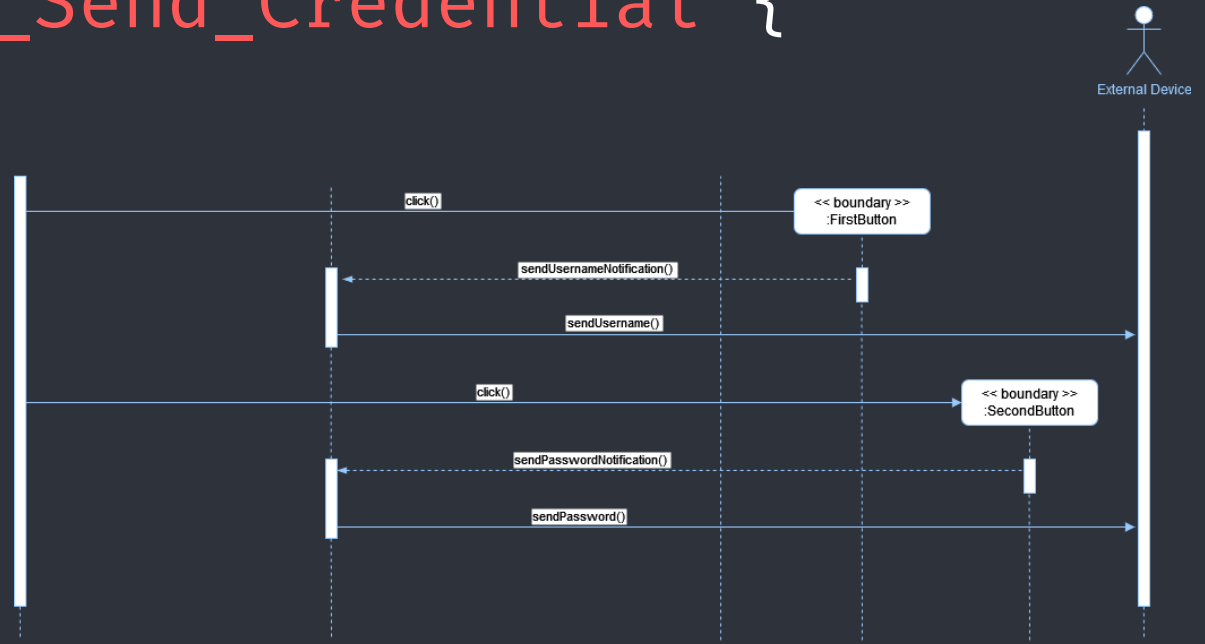
}

Arduino Language

PassChain.ino

Details.ino

SD_Send_Credential {



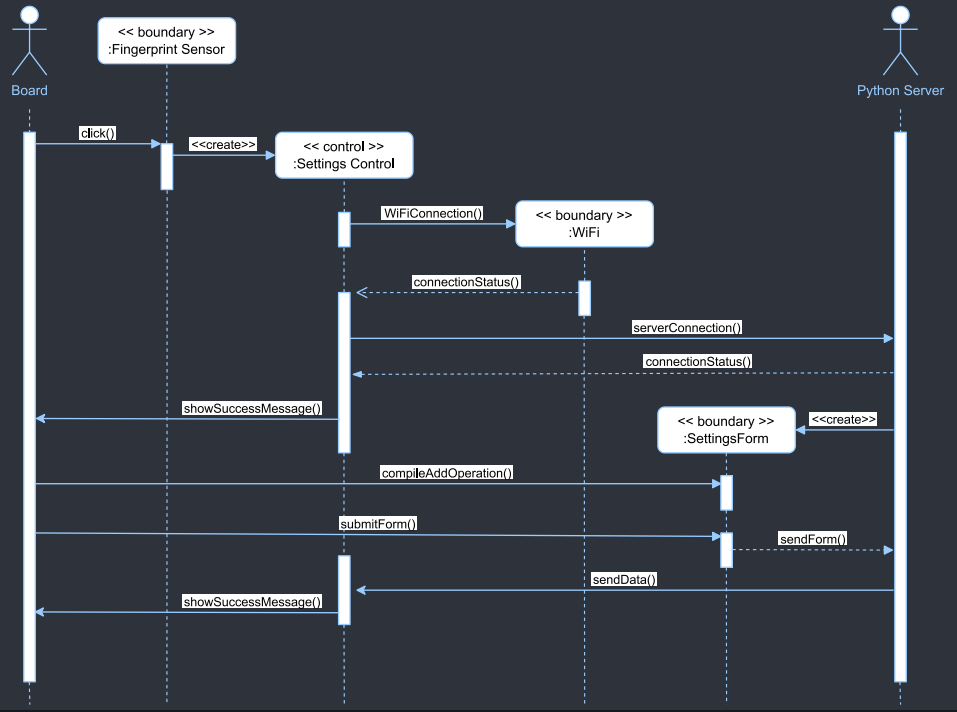
}

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 () {
```

< The board has an integrated 0.96'' OLED display, two programmable buttons and one for restarting, a power connector for batteries and a type-C port for connecting devices and powering the board itself. It has a memory of 16 Mb with SPIFFS as file system >

```
}

void Fingerprint () {
```

< Capacitive fingerprint sensor allows to authenticate the user. The sensor ensures the uniqueness of the user. The sensor can store 126 fingerprint: one of them is used to go back to the settings >

```
}
```

Arduino Language

PassChain.ino

Details.ino

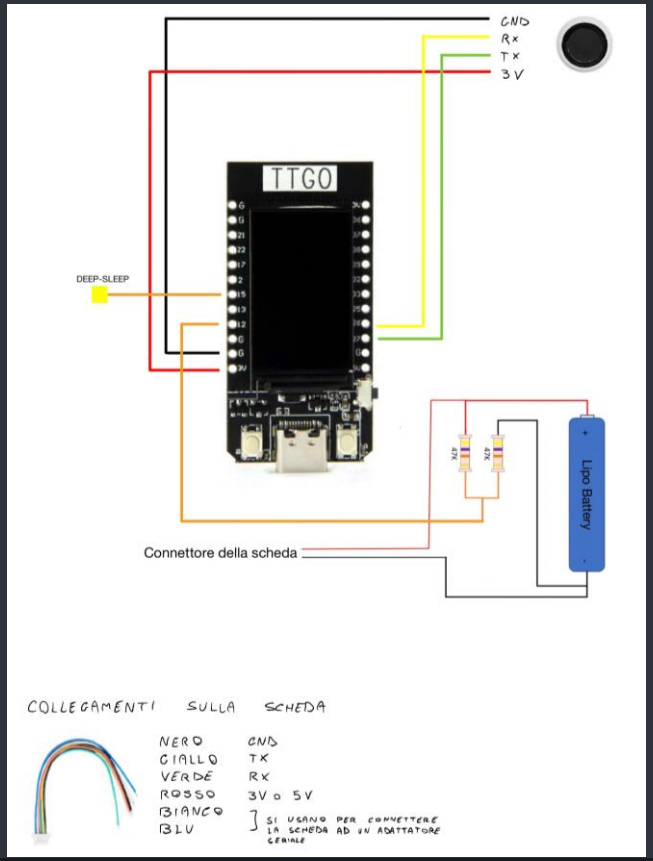
```
void Lora_Ttgo_Esp32 () {

    
    < LiPo battery 1 cell of 500 mAh 3.7 V allows
    to have a portable and independent device >

}
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: metal plaque to
8        wake up the device when it
9        goes in deep sleep after a
10       constant time
11
12     • GPIO 12: battery circuit
13
14 );
```

Details.ino



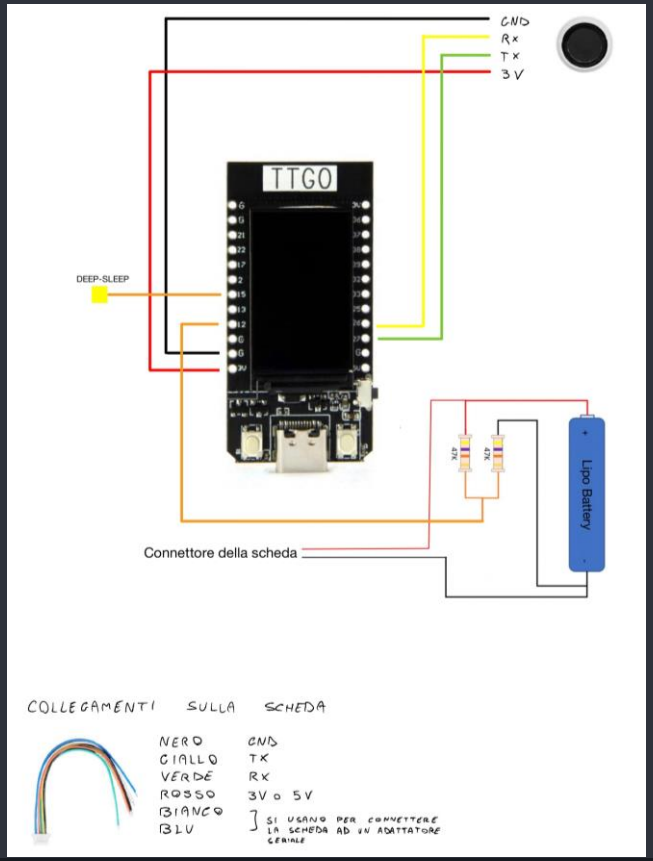
Arduino Language

PassChain.ino

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

- G: connection with GND
- Red line is connected to 3V
- GPIO 26: TX on the board
- GPIO 27: RX on the board

Details.ino



Arduino Language

```
03 {

    [Software]

        /* Libraries used, SSL communication,
           server Python */

}
```

```
Libraries_used(bool external) {

    if (!external) { /* details of Arduino's libraries */

        < TFT_eSPI.h >      * Arduino libray used to write text or
                            * images on display OLED.

        < SD.h >            * Arduino library to read and write on
                            * memory. The file system used is SPIFFS.

        < ArduinoJson.h >   * Arduino library used to manage JSON
                            * file in which stored user and HotSpot
                            * credentials
```

PassChain.ino

Details.ino

```
    < WiFiClientSecure.h >
    < mbedtls/gcm.h >
} else { /* details of external libraries */
    < BleKeyboard.h >
    < Adafruit_Fingerprint.h >
}
}
```

* Arduino libray used to create and setup SSL communication
* Used to encrypt and decrypt credentials with cipher AES - 128 bit GCM mode
* Used to simulate a virtual keyboard and to send messages with Bluetooth
* Used to enroll and delete user's fingerprints

Arduino Language

PassChain.ino

Details.ino

SSL_Communication {

```
/*****************************************************************
 * In this case, I create a secure communication between my Esp32 and  *
 * the external device that works as server to setup board's data.     *
 *****************************************************************/
```

Create Certificates

< CA and client certificates with the key associated to the client >

Store certificates

< The cerificates are stored in Arduino code and send to server >

Start Server Python

< Load server certificate and verify it and client certificate >

Receive datas

< Server creates a socket and receives data by Esp32 >

}

Arduino Language

PassChain.ino

Details.ino

Server_Python {

```
/*****************************************************************
 * It allows to setup Esp32 data. The operations concern: Add, Update, *
 * Delete, Setup Fingerprints, Setup Hotspot credentials          *
 *****************************************************************/
```

Start server

< welcome string, number of fingerprints, user and hotspot JSON credentials >

CRUD operations

< To setup Esp32, we can run CRUD operations on data: hotspot and credentials >

Setup Fingerprints

< Server verify if exist at least two FP: one to confirm operation and one to come back to settings in Esp32 >

Setup Hotspot

< Server can modify the SSID and password hotspot >

}

Arduino Language

```
04 {
    [Vulnerabilities &
     Security]

      /* Attacks and Security
      Solution */
}
```

PassChain.ino

Details.ino

```
1  void MITM_attack (Adv, Alice, Bob) {
2
3      /*****************************************************************
4       * PassChain is vulnerable to MITM attack about Bluetooth because  *
5       * it misses authentication between two devices. One solution is   *
6       * to use BLE v4.2 BR/EDR using ECDH based encryption and using    *
7       * the numeric compare binding model for authentication.           *
8       *****************************************************************/
9          < This solution is not strong because v4.2 to 5.0 have a vulnerability
10         called "BLURtooth" which exploits CTDK >
11
12       < This detection has been protected by the
13         GATT protocol available in the latest
14         versions of devices equipped with BLE >
   }
```

Arduino Language

PassChain.ino

Details.ino

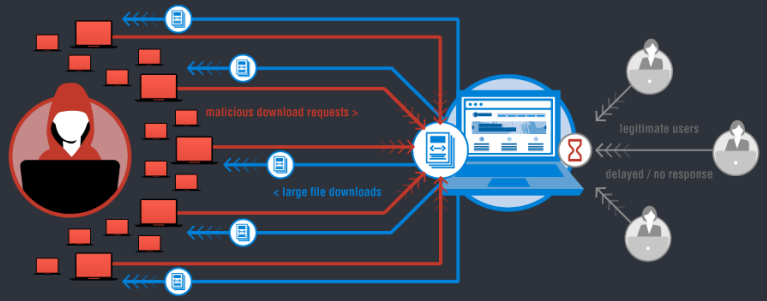
```
void Exploit_FP (memory) {

    /*****************************************************************
     * PassChain use Fingerprint to authenticate when user want run  *
     * some operation.                                               *
     *****************************************************************/

        < The vulnerability consists in UART exploitation because this
        sensor uses TTL Serial to communicate. For example, we can use
        Attify Badge to emulate a serial connection to access the target
        device >

      < A solution is to isolate PassChain from this device, in
        particular, user must not leave the device unattended >
}
```

Arduino Language

PassChain.ino

Details.ino

```
1  void DoS_attack (resources) {
2
3
4      /*****************************************************************
5       * User can do setup of PassChain connecting the board to the    *
6       * server Python.                                                *
7       *****************************************************************/
8
9          < A malicious can do a DoS attack exhaust the host's resources that run
10         server disrupting services of a host connected to a network >
11
12
13
14     < In this way the user cannot
         connect to server and to do the
         setup of the board >
   }
```



Arduino Language

PassChain.ino

Details.ino

```
void Security_Solution (crypto, SSL) {

    < The security solutions adopted are cryptography and
    SSL communication >


    Tag AES_128 (plainText,
                 key, IV);


        void SSL (certs);

}
```

* The user's credentials are stored
* encrypted using the AES-128 bit
* in GCM mode

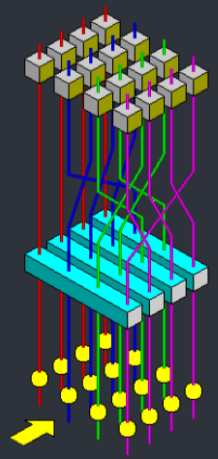
* Data are sended by server Python to
* Esp32 in clear because the
* communication are made by SSL that
* allows confidentiality, integrity and
* authentication

Arduino Language

PassChain.ino

Details.ino

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

    /*****************************************************************
     * AES-GCM is a block cipher provides authentication, confidentiality *
     * and integrity.                                                *
     *****************************************************************/

        < We have 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) {

    /*****************************************************************
     * The key must be generated randomly, for example by implementing    *
     * PUFs in Esp32.                                                     *
     *****************************************************************/

        < This is useful because the PUF is used to create keys that are
        generated on-demand and instantaneously erased once used. The PUF
        key value never exists in digital form within the circuitry of the
        security IC and the key is derived and produced on-demand from
        physical characteristics of circuit elements, they are never present
        in the device's nonvolatile memory >

     < In our case we have a hard-coded non-random
       test key >
}
```

Arduino Language

PassChain.ino | Details.ino

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

    /*****************************************************************
     * The IV are generated casually by Esp32 RNG Hardware.          *
     *****************************************************************/

        < The function esp_random() produces true random numbers if
        Wi-Fi or Bluetooth are enabled. The function returns an integer
        of 32 bit, so the string IV is builded in append by generating
        more TRNs to output a 96-bit string (12 byte) >

     < Every credential is own IV stored in clear
       and IV associated changes at each new
       encryption of the same credential >

}
```

IV

Arduino Language

PassChain.ino

Details.ino

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

- Step 01 It is a fast algorithm
- Step 02 The key can be 128 – 192 – 296 bit long not allowing brute force attacks
- Step 03 The block size is 128 bits
- Step 04 GCM mode allows authentication, confidentiality and integrity

```
);
```

Arduino Language

PassChain.ino

Details.ino

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

Step 01 In the case of nonce reuse both integrity and confidentiality properties are violated

Step 02 Short tags produce message forgeries

Step 03 GCM implementations are vulnerable to timing attacks

Step 04 GCM is vulnerable against cycling attacks

```
);
```

Arduino Language

PassChain.ino

Details.ino

```
void SSL (certs) {

    /**************************************************************************
     * SSL was used to create a secure channel during data exchange between *
     * the Python server and the board.                                       *
     **************************************************************************/

        < SSL allows us to authenticate the PassChain boards through the use
        of certificates. In fact, before the data exchange takes place, the
        server checks the client and server certificates and then the
        exchange takes place >

    < The disadvantages are that it uses a heavy protocol
      due to the continuous encryption and decryption of data
      when it departs and arrives at destination and to
      generate new certificates for every new board that
      connects to the server >

}
```

Arduino Language

PassChain.ino

Details.ino

```
Serial.println ('SSL Advantages: ' +
```

Authentication — Client uses the server's public key to encrypt the data; the server uses the public key in the client certificate to decrypt the data the client sends

Integrity — SSL provides data integrity by calculating a message digest

Confidentiality — SSL uses a combination of symmetric and asymmetric encryption to ensure message privacy

```
);
```

Arduino Language

PassChain.ino

Details.ino

```
1
2
3
4   void thank_you () {
5
6
7       Serial.println('Thank you');
8
9
10      Serial.println('For your Attention');
11
12  }
13
14
```

Arduino Language