Università degli Studi di Firenze

Scuola di Ingegneria - Dipartimento di Ingegneria dell'Informazione

---

Tesi di Laurea Triennale in Ingegneria Informatica

Studio e implementazione di una soluzione per mutua autenticazione in ambito IoT device embedded

Candidato
Samuele Meta

Relatore
Prof. Paolo Nesi

Correlatore
Dott. Angelo Difino

Anno Accademico 2017/2018

Indice

Ringraziamenti

Innanzitutto, un primo, doveroso, ringraziamento è da rivolgersi al Prof. Paolo Nesi, responsabile di avermi proposto un elaborato di notevole interesse, nonché di avermi guidato in questo percorso. Desidero, inoltre, ringraziare il mio correlatore, il Dott. Angelo Difino, per tutto il tempo dedicatomi e la cordialità che lo ha sempre contraddistinto.

Un caloroso ringraziamento va ai miei genitori, senza i cui sacrifici e insegnamenti non avrei raggiunto questo traguardo. Non potrò mai essere sufficientemente grato per tutti i piccoli gesti di affetto e per l'estrema fiducia riposta nelle mie decisioni. Mi auguro che quanto conseguito possa renderli orgogliosi di me. Desidero ringraziare anche mio fratello, sempre capace di strapparmi un sorriso, soprattutto nelle giornate più pesanti.

Un ringraziamento va anche a tutti i parenti che in questi mesi si sono interessati al mio lavoro, non mancando mai di farmi sentire il loro appoggio. Che si sia trattato di una telefonata o un semplice messaggio, mi commuove essere consapevole di poter sempre contare su di loro, nonostante la distanza che ci separa.

Ringrazio tutti i colleghi con i quali ho avuto il piacere di condividere la mia quotidianità negli ultimi tre anni. È soprattutto merito loro se conservo un ricordo positivo di questa esperienza universitaria, se le delusioni sono state meno amare e le gioie più autentiche. In particolare, desidero ringraziare Gianluca Botteri, che non ha mai mancato di offrirmi il suo sostegno anche nelle circostanze più avverse.

Un insolito ringraziamento va anche a `Stack Overflow`, per avermi sempre fornito le risposte che cercavo, eccezion fatta per la tesi. Ciò mi ha permesso di scoprire risorse che non pensavo di possedere, ma anche di comprendere cosa significhi lasciare un primo, piccolo, segno nel mondo dell'informatica. Ed è per emozioni come questa che vale la pena impegnarsi nello studio di ciò che si ama.

Questo elaborato conclude un capitolo della mia vita e corona un percorso intenso, ma ricco di soddisfazioni. Desidero, pertanto, rivolgere un ringraziamento finale a chiunque vi abbia, anche solo in minima parte, contribuito.

Introduzione

Nel seguente elaborato di tesi si vuole discutere il processo di integrazione del paradigma di mutua autenticazione in una comunicazione client-server in ambito Internet of Things (IoT). Il server preso in esame è la piattaforma cloud `Snap4City`, sviluppata presso il Distributed Systems and Internet Technologies (DISIT) Lab dell'Università di Firenze. I client impiegati sono rappresentati dai chip WiFi `ESP32` ed `ESP8266`, montati rispettivamente sullo `Snap4City IoT Button` e `Arduino Uno`.

La finalità di questa trattazione è quella di mettere in luce i requisiti necessari alla realizzazione di tale modello, la sua attuazione e i considerevoli benefici che ne possono scaturire. Particolare enfasi verrà posta nella discussione della sicurezza, tema quanto mai centrale nel mondo IoT. Come si evince dal report annuale diffuso da Eclipse, questa rappresenta ancora la maggiore preoccupazione degli addetti ai lavori, superando considerevolmente tematiche quali *Data Collection & Analytics* e *Connectivity* [1].

Nel primo capitolo vengono brevemente definiti alcuni concetti propedeutici alla comprensione dell'esposizione successiva. In principio viene fornita una panoramica sul mondo dell'IoT e le annesse problematiche di sicurezza. Successivamente, viene presentata la piattaforma cloud `Snap4City`, con spe-

cifico riguardo per le caratteristiche dell'`Orion Broker`. In ultimo, è descritto il protocollo crittografico Transport Layer Security (TLS), focalizzandosi sulla sua implementazione in mutua autenticazione e il relativo utilizzo di certificati e chiavi.

Nel secondo capitolo si propone l'analisi di un client basato su `Arduino Uno` e lo shield WiFi `UART`, montante il chip `ESP8266`. Nella prima sezione sono discusse le specifiche hardware delle componenti in esame e la loro integrazione. In seguito, viene descritto un concreto scenario di interesse e il conseguente comportamento del dispositivo durante l'esecuzione. In conclusione, viene argomentata la soluzione software sviluppata, mostrandone le peculiarità e le maggiori problematiche riscontrate.

Nel terzo capitolo viene trattato come il lavoro precedentemente svolto sia adattabile, previa qualche modifica, al chip `ESP32`, microprocessore dello `Snap4City IoT Button`. Nella prima parte viene presentata la nuova soluzione hardware, più compatta rispetto allo shield `Arduino`. Nella parte centrale è riportata la macchina a stati illustrante i principali scenari di interesse. Infine, vengono evidenziate le modifiche apportate a livello software, riconducibili alla mancata compatibilità delle librerie.

Capitolo 1

Preliminari

1.1 Internet of Things

Con il termine Internet of Things (IoT) si fa riferimento a un sistema di interconnessione tra calcolatori, oggetti, animali o persone a cui è stato fornito un univoco identificativo e la capacità di trasferire dati sulla rete [2]. Questo è costituito, in generale, da un insieme di dispositivi che fanno uso di processori integrati, sensori e strumenti di comunicazione per collezionare, scambiare ed elaborare i parametri che acquisiscono dal loro ambiente. Ciò avviene, nella maggior parte dei casi, senza alcuna interazione con l'uomo, sebbene gli sia riservata la facoltà di apportare regolazioni e accedere ai dati. Più di frequente, invece, questi vengono trasmessi tra i dispositivi stessi, con la finalità di arricchire la propria percezione dello scenario operativo. Le loro rilevazioni, in seguito, possono essere inoltrate a una piattaforma cloud, dove vengono aggregate e analizzate per trarne diverse conclusioni di interesse. La premessa di base dell'IoT, nonché il suo scopo ultimo, è rappresentato, pertanto, dal connettere tutto ciò che ancora non ha accesso ad Internet, permettendogli di diventare protagonista attivo del dialogo sopra

citato. Tale integrazione, sempre più marcata, tra il mondo fisico e quello delle macchine, consente notevoli margini di miglioramento nell'area dell'automazione, ottimizzandone l'accuratezza ed efficienza. Un'ampia varietà di industrie se ne sta avvalendo per posizionarsi in modo più competitivo sul mercato, accrescendo la qualità dei servizi offerti ai clienti finali e avvalendosi di strumenti di *decision-making* più performanti. La vertiginosa ascesa di questo trand non coinvolge però solo la sfera industriale, bensì un crescente numero di settori produttivi, tanto che nel 2017, Gartner, società leader nell'ambito della ricerca e analisi nel campo dell'Information Technology, riportava che alla fine dello stesso anno circa 8.4 miliardi di *oggetti* connessi alla rete sarebbero risultati attivi [3]. Inoltre, in un ulteriore report, prevedeva che tale cifra sarebbe inesorabilmente aumentata, fino a raggiungere la soglia dei 20 miliardi entro il 2020 [4]. Dalla lampadina wireless agli elettrodomestici intelligenti, l'IoT sta indubbiamente permeando progressivamente la quotidianità, andando a ridefinire anche le abitudini più comuni.

Basti pensare ai microinfusori, noti anche come pompe di insulina, che venivano prescritti nella terapia del diabete già dai primi prototipi del 1963, rendendo possibile una costante iniezione dell'ormone nel corso della giornata. Sebbene il loro scopo iniziale fosse unicamente quello di regolare il livello di glucosio nel sangue, ad oggi rappresentano un eccellente esempio delle potenzialità dell'IoT. La pompa, infatti, oltre alla sua funzione primaria, consente di visualizzare l'evoluzione temporale dei parametri monitorati, di memorizzarli e di prendere decisioni basandosi sui loro valori. Ad esempio, se per mezzo degli ultimi campioni verrà prevista una successiva misurazione al di sotto di una determinata soglia, la somministrazione potrà essere sospesa in attesa di una situazione più stabile. Tale scenario può essere ulteriormen-

te ampliato, introducendo una piattaforma cloud a cui far pervenire i dati rilevati, ingrediente principale per consentire un'ampia gamma di operazioni:

- i valori di battito cardiaco e glucosio nel sangue possono essere riportati contemporaneamente in un grafico, mettendo in luce l'influenza delle condizioni di stress sulla concentrazione di zuccheri.
- le informazioni raccolte possono essere schematizzate in un file *Excel* per generare report periodici da inoltrare al medico di fiducia.
- le rilevazioni possono essere sincronizzate con il calendario, per determinare le attività più provanti nel corso della giornata.
- si possono applicare algoritmi di intelligenza artificiale per generare nuovi modelli predittivi.
- per prevenire eventuali crisi, può essere realizzato un sistema di notifica capace di avvisare un contatto di emergenza qualora le ultime misurazioni destino preoccupazione.
- tutti i rilevamenti possono essere inviati al proprio *wearable* per una rapida visualizzazione e ricevere eventuali riscontri acustici [5].

Il ogni caso, tuttavia, è fondamentale che l'informazione scambiata con la piattaforma cloud rimanga accessibile esclusivamente al suo proprietario. Infatti, se si prendono in esame anche il consolidato impiego nell'ambito delle videocamere di sicurezza o le iniziali sperimentazioni nel settore delle serrature, si può realizzare quanto la sicurezza e confidenzialità di questi dati personali sia cruciale. Questa si rende ancor più manifesta passando dalla sfera privata a quella pubblica, in cui è possibile, ad esempio, ravvisare l'utilizzo di sensori per rilevazioni militari. Anche innovazioni prettamente

positive, quali le auto a guida autonoma, dovranno scongiurare che il sistema di intercomunicazione possa essere compromesso e sfruttato per fini terroristici. Sebbene ciò possa far presagire una grande sollecitudine nell'attuazione di nuove soluzioni sicure, lo scenario attuale è, invece, agli antipodi. Emblematico, in tal senso, è l'ironico tweet "*The S in the IoT stands for Security*" [6], pubblicato dallo sviluppatore Oracle Oleg Šelajev. Uno studio condotto dall'azienda Symatec, operante nell'industria della sicurezza informatica, mostra, infatti, come nel corso del 2017 sia stato rilevato un aumento del 600% degli attacchi a dispositivi IoT [7]. Kaspersky, attiva nel medesimo ambito, soltanto nella prima metà del 2018 ha riscontrato, invece, una triplicazione delle tipologie dei malware destinati agli *oggetti* [8]. Facilitati da standard di sicurezza ancora non all'altezza, tali *exploit* hanno permesso il proliferare di *botnet* atte a condurre attacchi DDoS, nonché di sfruttare numerosi dispositivi per il *mining* di criptovalute.

1.2 Piattaforma Cloud Snap4City

La piattaforma cloud `Snap4City` è un progetto open source volto a realizzare soluzioni flessibili per monitorare lo stato della città e la sua evoluzione, sfruttando algoritmi e tecnologie all'avanguardia nel settore dell'intelligenza artificiale e dell'analisi dei *big data*. Essa permette, inoltre, l'integrazione di molteplici tipologie di dispositivi IoT, proponendosi come valido supporto anche per progetti in ambito industria 4.0 [9]. Tra i numerosi servizi offerti si annoverano la possibilità di registrare e gestire i propri sensori, la capacità di generare *dashboard* dedicate e personalizzabili e la facoltà di implementare applicazioni incentrate sui dati in ingresso. Nel capitolo conclusivo si avrà modo di illustrare come il lavoro presentato in questa trattazione si innesti

nel quadro generale appena delineato. In primo luogo, tuttavia, è bene soffermarsi su una sua componente che opera in modo trasparente all'utente, ma quantomai importante per la corretta comunicazione con i dispositivi: il Broker. Esso agisce da intermediario e il suo scopo è quello di fornire al destinatario, ovvero la piattaforma cloud, l'informazione secondo un formato standardizzato, indipendentemente dai diversi sensori che vi possono comunicare. Ciò comporta dei notevoli vantaggi, tra cui:

- il mittente e il destinatario non interagiscono mai in modo esplicito e diretto, rendendo più lasco l'accoppiamento che si instaura tra i due. Ciò, inoltre, incrementa la sicurezza della trasmissione, impedendo che uno dei sensori possa essere attaccato a partire dalla piattaforma.

- è possibile ricorrere al *device shadowing*, corrispondente alla copia dell'ultimo stato riportato dal dispositivo. Se per qualche ragione questo non fosse più disponibile, il server potrà comunque continuare ad interrogare la sua copia, ricevendo come risposta sempre la stessa informazione, ma senza incorrere in eccezioni.

Inoltre il Broker rende possibile la gestione dell'intero ciclo vitale delle informazioni, permettendo l'esecuzione di query, registrazioni e sottoscrizioni. In questo elaborato si farà riferimento a Orion, implementazione delle REST API NGSI9/10 facente parte della piattaforma `FIWARE`. Il punto focale del suo modello è rappresentato dal concetto di *entità*, rappresentazione virtuale di qualsivoglia tipologia di oggetto del mondo reale. Ad essa possono essere assegnate un *tipo* e un *identificativo*. Nello scenario illustrato in Figura 1.1, l'entità è contraddistinta dalla tipologia `Ambiental` ed è univocamente determinata dall'ID `myArduino`. Ogni sua proprietà viene espressa in termini di *attributi*, a loro volta caratterizzati dai parametri precedenti. Nell'esem-

```
{ "contextElements": [ {
   "type": "Ambiental",
   "isPattern": false,
   "id": "myArduino",
   "attributes": [ {
     "name": "temperature",
     "type": "float",
     "value": "23"
    }, {
     "name": "humidity",
     "type": "float",
     "value": "72"
    } ] } ],
 "updateAction": "APPEND"
}
```

Figura 1.1: Esempio di richiesta JSON inoltrata al Broker.

pio sono riportati due attributi, temperatura e umidità, entrambi di tipo `float`. Eventualmente, questi possono essere raggruppati secondo categorie concettuali. Il campo `isPattern` viene, invece, regolato a `true` solo in fase di ricerca, qualora sia necessario fare uso delle *regular expressions*. Nel complesso, la struttura dati che racchiude l'informazione riguardante un'entità è detta *context element*. Questa è costruita secondo una lista di triple ottenute mediante l'applicazione dei principi precedenti. Infine, la richiesta si conclude con il valore della *action*, per distinguere se effettuare un *update* dell'infromazione o un suo *append* [10].

1.3 Transport Layer Security

Transport Layer Security (TLS) è un protocollo crittografico designato a rendere sicure connessioni basate su infrastrutture che non lo sono. Un suo corretto utilizzo garantisce di comunicare con il corretto server di un arbitrario servizio su Internet, salvaguardando lo scambio informativo da attacchi esterni [11]. Introducendo il paradigma della mutua autenticazione, anche il client è tenuto a fornire al suo interlocutore un certificato digitale per definire la propria identità. Ciò permette al server di prevenire molteplici vulnerabilità associate ad un approccio basato sulla coppia `username` e `password`, quali tentativi di *phishing* e di *keylogging*. Il processo di autenticazione e creazione di un canale criptato passa attraverso le seguenti fasi, rappresentate in Figura 1.2.

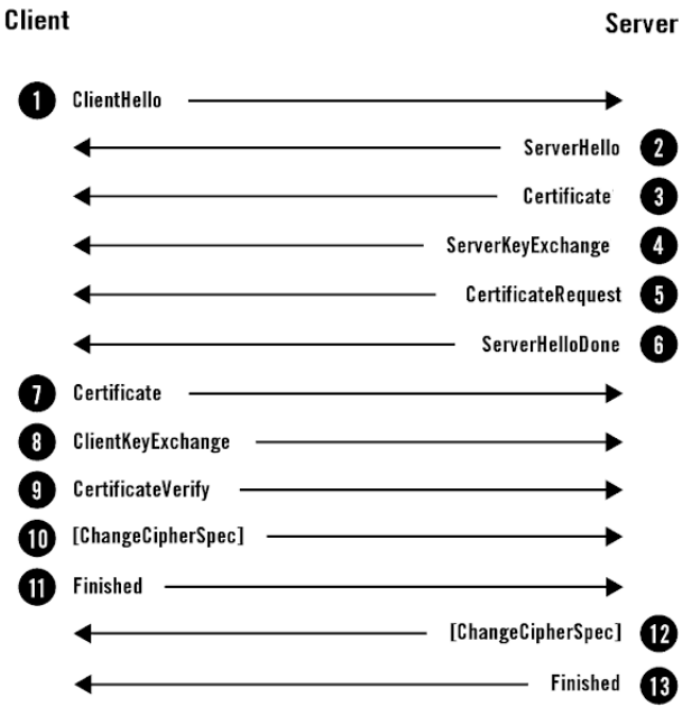


Figura 1.2: TLS handshake nelle sue fasi costitutive.

- Il client richiede l'accesso a una risorsa protetta tramite il messaggio di `ClientHello`, in cui specifica alcune opzioni quali la versione del protocollo in uso e i cifrari da lui supportati. A questa comunicazione il server risponde con il messaggio di `ServerHello`, dove sono riportati i parametri selezionati tra quelli proposti dal mittente.

- Il server inoltra il messaggio `Certificate`, nel quale è contenuta la catena di certificazioni X.509 secondo un *encoding* ASN.1 DER, e il messaggio `CertificateRequest`, per richiedere la medesima operazione al client. Informazioni addizionali, riguardanti la chiave di cifratura, saranno riferite mediante il messaggio `ServerKeyExchange`, qualora la *cipher suite* negoziata lo richiedesse. Una volta effettuate queste operazioni, il server restituisce il controllo al client tramite il messaggio `ServerHelloDone`.

- Il client provvede a generare il messaggio di `ClientCertificate`, nel quale allega la propria certificazione o segnala la sua assenza mediante un *alert*. A questo segue il suo contributo allo scambio della chiave, racchiuso nel messaggio di `ClientKeyExchange`. Per dimostrare al server la propria autenticità, un hash delle trasmissioni precedenti viene inglobato nel messaggio `CertificateVerify`. Infine, stabilito il cifrario in vigore tramite messaggi `ChangeCipherSpec`, conclude la sua fase di inoltro con il messaggio `Finished`, che sancisce l'inizio della comunicazione criptata.

L'implementazione della TLS *handshake* completa, secondo i precetti descritti, è stata realizzata avvalendosi della libreria `WiFiClientSecure` [12] [13]. Il suo utilizzo verrà trattato diffusamente nei capitoli successivi, mostrandone i parametri richiesti e l'impatto sui dispositivi in esame.

1.4 Certificati e Chiavi

La crittografia *asimmetrica* implementa un approccio in cui, ad ogni attore coinvolto nella comunicazione, viene assegnata una coppia di chiavi. La prima, nota come chiave *pubblica*, viene resa accessibile a chiunque, mentre la seconda, detta chiave *privata*, deve rimanere segreta. In virtù della relazione matematica che sussiste tra le due e le rende dipendenti, queste possono essere adoperate per effettuare la codifica e decodifica dei messaggi da scambiare sulla rete. Infatti, se l'informazione viene cifrata tramite una chiave pubblica, solo compiendo l'operazione inversa, avvalendosi della corrispondente chiave privata, si potrà comprendere il contenuto originale.

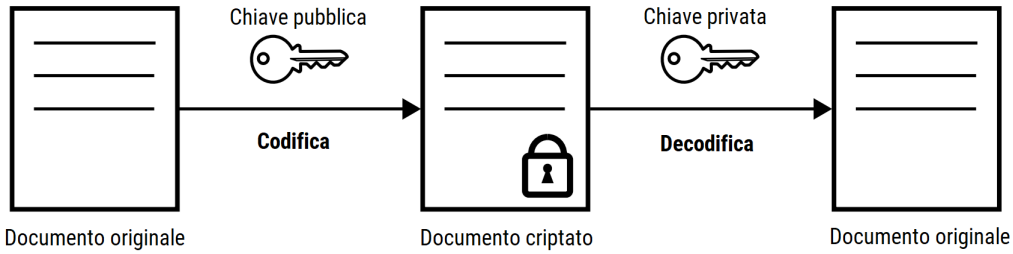


Figura 1.3: Codifica e decodifica di un documento mediante chiavi.

Viceversa, applicando la chiave privata all'informazione di partenza, si ottiene una nuova formulazione criptata, che sarà possibile tradurre grazie alla chiave pubblica. Dal momento che questa è accessibile da chiunque, tale procedura viene sfruttata per fornire una *signature*, ovvero un'univoca firma digitale. In entrambi i casi si è fatto ricorso all'algoritmo RSA, rappresentante la tecnica maggiormente affermata per conseguire una crittografia asimmetrica.

Un *certificato* è un documento che contiene una chiave pubblica, informazioni riferite all'ente a cui è associato, il periodo di validità e una propria signature. Tale struttura, spesso formalizzata secondo lo standard `X.509`,

è un caso particolare del linguaggio astratto `ASN.1` (Abstract Syntax Notation One), un insieme di regole che supportano la diffusione, il trasporto e lo scambio di complesse strutture dati. Dal momento che esse descrivono unicamente il contenitore ma non il contenuto, al suo interno sono supportati molteplici tipologie di *encoding* dell'informazione. Analizziamo nel dettaglio i due formati che ricorreranno più avanti nella trattazione:

- PEM (Privacy Enhanced Mail), rappresenta il certificato e la chiave secondo una stringa racchiusa nelle guardie di `BEGIN` ed `END`. I caratteri ASCII esprimono il risultato dell'*encoding* base64 dei dati binari.
- DER (Distinguished Encoding Rules), rappresenta il certificato e la chiave secondo la loro codifica binaria, senza che vi sia inclusa alcuna clausola di inizio o fine.

Indipendentemente dal formato utilizzato, il parametro che contraddistingue la sicurezza delle credenziali è la lunghezza della chiave. All'aumentare del numero di bit di cui è composta, infatti, corrisponde un considerevole incremento della complessità di fattorizzazione, operazione che, se portata a termine, permette di identificare l'informazione riservata. Per conseguire un soddisfacente livello di protezione, in accordo con le proprietà sopra citate, si è optato per i seguenti criteri:

1. il certificato lato server è stato generato servendosi di una chiave a 2048 bit. Sebbene queste stiano progressivamente lasciando spazio a quelle a 4096, assicurano ancora un livello protettivo più che soddisfacente. Ciò è confermato dal fatto che grandi aziende, quali Github e Mozilla, continuino ad affidarvisi. Nello scenario in esame, inoltre, chiavi a 4096 bit sarebbero state superflue, dal momento che avrebbero comportato

un notevole appesantimento del carico sulla CPU e un rallentamento nella fase di *handshake* [14].

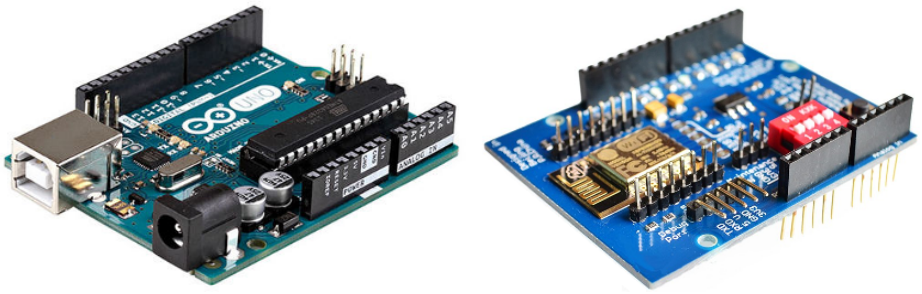
2. il certificato lato client, invece, è stato generato avvalendosi di una chiave a 1024 bit. Per comprendere le motivazioni di tale scelta, bisogna tener presente che la progettazione è avvenuta in un contesto *embedded*, dove i requisiti di memoria sono particolarmente stringenti. Inoltre, è bene ricordare che il record di fattorizzazione di una chiave, ad opera del gruppo capitanato dal professor Lenstra, è fermo alla lunghezza di 768 bit dal 2009. Lo stesso Lenstra ha dichiarato che replicare la medesima procedura per chiavi a 1024 bit comporterebbe una difficoltà maggiore di tre ordini di grandezza [15]. Sebbene ammonisca che nel prossimo futuro 1024 bit potrebbero non essere sufficienti, ad oggi rappresentano un ottimo compromesso tra la RAM del dispositivo e la sicurezza acquisita.

Capitolo 2

Arduino Uno ed ESP8266

2.1 Specifiche hardware

`Arduino UNO` è una scheda open-source sviluppata da Arduino.cc, basata sul microcontrollore `ATmega328P`. Essa si presenta equipaggiata di 14 pin di input/output digitale, 6 pin analogici, un cristallo di quarzo a 16MHz e una memoria flash da 32KB [16]. Grazie al suo ingresso USB, affiancato a quello dell'alimentazione, è stato possibile programmarne la logica di funzionamento mediante il software omonimo.



(a) Arduino Uno (b) Shield WiFi

Figura 2.1: Dettaglio delle principali periferiche hardware utilizzate.

Sopra di essa è stato installato lo shield WiFi `ESP8266 ESP-12E UART`, responsabile della connettività Internet via WiFi e degli scenari ad essa riconducibili. Il chip `ESP8266` è caratterizzato da un processore RISC L106 a 32 bit, basato su Tensilica Xtensa Diamond Standard 106Micro, operante a 80 MHz. La RAM adibita al segmento dati ammonta a 96 KiB, mentre quella relativa alle istruzioni di programma si attesta sui 64 KiB; nel suo complesso, la memoria *flash* esterna raggiunge la capacità di 1 MiB. Più avanti nella trattazione, si avrà modo di mettere in luce come queste proprietà abbiano rappresentato le più grandi limitazioni ai fini della realizzazione del progetto. Infine, gli standard di trasmissione compatibili coprono le varianti `IEEE 802.11 b/g/n`. Una volta integrato il modulo, si è potuto, grazie al suo *switch*, abilitare e disabilitare la scrittura verso di esso e `Arduino`, senza la complicazione di doverli dividere. Inoltre, una sua specifica configurazione ha reso possibile l'attuazione della comunicazione seriale tra le due componenti.

(a) Sensore DHT22 (b) Pulsante (c) Led RGB

Figura 2.2: Dettaglio delle componenti aggiuntive.

Per permettere l'interazione con l'utente finale, al `PIN 4` è stato associato il pulsante esterno, rappresentante l'unico input diretto. Ai fini di garantire un riscontro alle sue azioni, i `PIN 11`, `12` e `13` sono stati, di conseguenza, riservati ai tre canali del led RGB. Infine, al `PIN 7`, è stato associato il

sensore `DHT22`, artefice delle misurazioni di temperatura e umidità inoltrate al server.

2.2 Scenario di interesse

Prima di trattare più approfonditamente i dettagli implementativi, esaminiamo, facendo riferimento a Figura 2.3, uno scenario operativo, in cui il dispositivo viene utilizzato dall'utente per monitorare in tempo reale i valori di temperatura e umidità della propria abitazione.

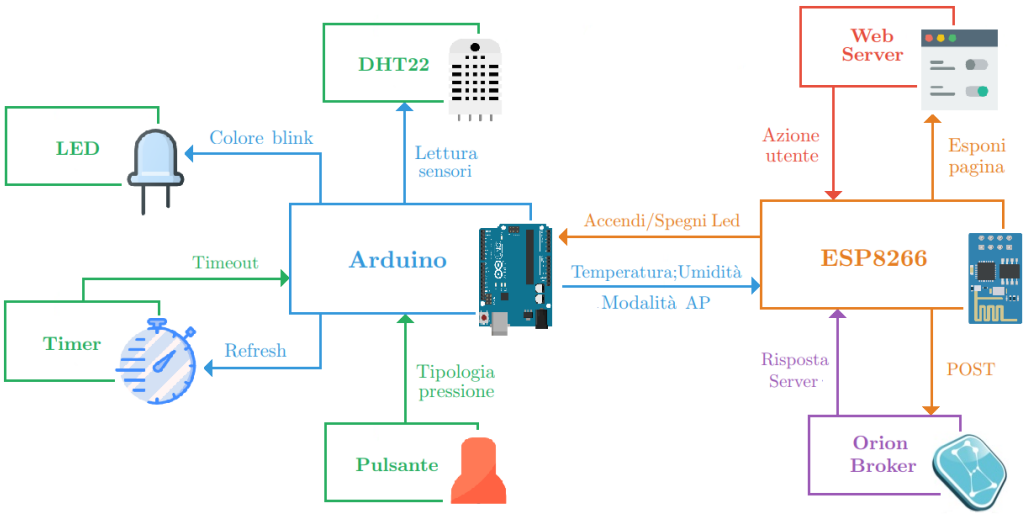


Figura 2.3: Struttura delle dipendenze tra le diverse componenti.

1. Se l'utente effettua una breve pressione del pulsante, tale evento scaturisce una lettura del sensore, i cui parametri verranno immediatamente notificati alla piattaforma cloud tramite un messaggio. Questa fase è contraddistinta dall'illuminarsi del led di colore blu, sinonimo dello svolgersi di tale operazione. Una volta conclusa, il suo esito sarà reso

noto mediante una luce verde, in caso positivo, o rossa, nell'eventualità che siano state riscontrate complicazioni di rete o credenziali errate.

2. Se l'utente esegue una pressione prolungata, veicolante la richiesta di attivare l'Access Point, la tonalità bianca del led suggerirà il corretto ingresso nella modalità di setup. Ciò gli permetterà di fornire nuove connessioni, eliminare qualcuna delle precedenti oppure eseguire un reset totale delle configurazioni immesse.

3. Dal momento che si desidera monitorare l'andamento temporale dei valori di temperatura e umidità indipendentemente dall'azione dell'utente, ciclicamente, con cadenza di 20 secondi, viene effettuata una nuova rilevazione poi trasmessa alla piattaforma cloud. Ciò non esclude, tuttavia, la possibilità di eseguire una misurazione *on demand*, come visto nel primo punto, in occorrenza della quale il timer verrà riavviato.

2.3 Programmazione Arduino

La scheda Arduino esegue una logica estremamente semplice e lineare, eppure fondamentale per il corretto funzionamento globale. Essa ha il compito di mettere in comunicazione i numerosi ingressi e uscite, mediando le loro richieste e vigilando sulle loro interazioni. Osserviamo più nel dettaglio l'implementazione del ciclo `loop`, nel quale sono racchiuse le istruzioni da riproporre in modo periodico:

- il primo controllo, adempiuto tramite la libreria `OneButton`, consiste nella lettura del `PIN 4`, terminale del collegamento con il pulsante esterno. In caso di riscontro positivo, viene individuata la tipologia della pressione - corta o prolungata - e invocata la funzione associata. Nell'e-

ventualità venga effettuato un singolo click, il microcontrollore provvederà ad eseguire il metodo `sendData()`. Questo coordina la lettura del sensore `DHT22`, per aggiornare i valori di temperatura e umidità salvati in passato, e l'invio della nuova informazione, mediante scrittura sulla seriale nella forma "`temperatura;umidità`". Se venisse, invece, individuata una pressione di qualche secondo, ad attivarsi sarebbe la funzione `sendLongPress()`, responsabile di notificare l'evento allo shield tramite l'invio del carattere `L`. La scelta di veicolare a singoli caratteri istruzioni concettualmente più complesse, ripresa anche in seguito, è stata realizzata per aumentare l'affidabilità del processo, nonché la sua velocità.

- il secondo controllo riguarda la quantificazione dell'intervallo temporale trascorso tra un'attività e la successiva. Essendo la pressione del pulsante un servizio *on demand* sporadico, il dispositivo dovrà provvedere autonomamente a contattare il server per fornirgli nuove rilevazioni. Questi messaggi verranno trasmessi ad ogni scadenza dell'arco di tempo prestabilito.

- l'ultimo accertamento consiste nella lettura della seriale, ai fini di verificare la potenziale presenza di messaggi provenienti dallo shield. Questi sono costituiti da singole lettere, maiuscole o minuscole, indicanti l'iniziale di uno dei colori supportati dal led. Nel primo caso si procederà ad illuminare il led secondo la tonalità associata, altrimenti il relativo canale verrà spento.

2.4 Programmazione ESP8266

2.4.1 ESP8266WiFi

Dal momento che `Arduino` supporta unicamente le operazioni basilari descritte nel paragrafo 2.3, occorre applicarvi lo shield per conferirgli la connettività necessaria al dialogo con la piattaforma cloud. I suoi differenti risvolti sono stati gestiti mediante le primitive esposte dalla libreria `ESP8266WiFi`, indispensabile per raggiungere il risultato citato. In principio si è dovuta implementare un'efficace alternanza tra le due principali modalità del dispositivo:

- la modalità **Station** (STA), in cui il microprocessore interagisce con la rete secondo il ruolo di *client*, potendo, pertanto, fare accesso a una connessione disponibile per inoltrare dati.
- la modalità **Access Point** (AP), in cui il microprocessore ricopre il ruolo di *server*. Esso espone una connessione privata, accessibile dall'utente grazie alle credenziali allegate al dispositivo, in cui saranno erogati i servizi descritti nel paragrafo 2.4.2. Per poter raggiungere le pagine di configurazione si dovrà contattare l'indirizzo IP `192.168.4.1`.

Essendo quest'ultima considerevolmente dispendiosa, particolare attenzione è stata posta nella corretta transizione verso la modalità STA. Ciò è fondamentale per scongiurare di incorrere in un reset del dispositivo, scaturito dall'elevata richiesta, non soddisfatta, di memoria.

In un secondo momento, dopo aver constatato che la libreria preposta a coniugare l'utilizzo di molteplici connessioni non era sufficientemente dinamica, si è provveduto a fornirne una personale interpretazione. Questa garantisce che il chip si connetta sempre alla rete dal segnale più intenso,

aggiornandosi ad ogni aggiunta o rimozione delle configurazioni in memoria. Per adempiere questo compito, si occupa di verificare quali `SSID` salvate risultino presenti nella lista delle connessioni individuate dall'ultima scansione. Dopo averle ordinate secondo il parametro `RSSI`, indice della loro robustezza, procederà, semplicemente, a progressivi tentativi di instaurare un collegamento.

2.4.2 ESP8266WebServer

Una volta regolata la duplice modalità WiFi, si è proceduto a creare un'interfaccia di configurazione per consentire l'immissione delle informazioni durante la fase di setup. A partire dalla libreria `ESP8266WebServer` è stato possibile istanziare l'oggetto `server`, a cui sono state assegnate le pagine che si è ritenuto opportuno rendere raggiungibili dall'utente. Il comportamento di ciascuna di esse, a sua volta, è stato implementato nella relativa funzione associata. Analizziamo più nel dettaglio gli eventuali scenari:

- se non viene rilevata alcuna preferenza precedentemente inserita, il server reindirizza l'utente alla pagina di prima configurazione. Tramite questa è possibile riportare le caratteristiche e proprietà del dispositivo, oltre a poter specificare una prima connessione WiFi a cui collegarsi. Inoltre, il duplice metodo di autenticazione garantisce all'utente la facoltà di determinare, autonomamente, il livello di sicurezza desiderato. Al momento del salvataggio viene invocata la funzione `handleSave()`, incaricata di trascrivere in memoria i dati in ingresso, dopo averne appurato la correttezza. Qualora un campo presenti un'informazione mancante o di lunghezza non consona, infatti, viene notificato l'esito negativo dell'operazione, invitando l'utente a correggere i parametri errati. Per prevenire questa eventualità e ridurre le responsabilità del

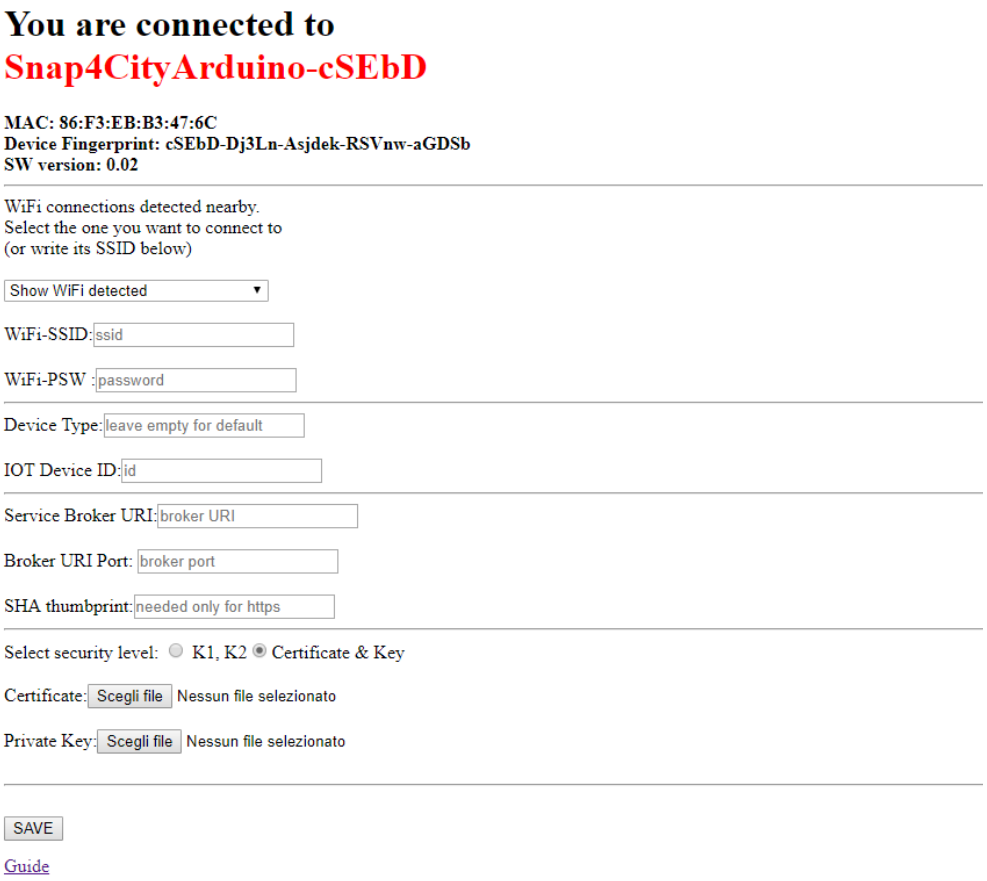


Figura 2.4: Interfaccia di prima configurazione.

chip, il controllo del formato del certificato e della chiave è stato demandato al browser. Ciò ha permesso di fare affidamento su una maggiore potenza di calcolo e, al contempo, di avvalersi della flessibilità del `Javascript`, rendendo possibile un feedback in tempo reale circa la validità del file caricato.

- se il dispositivo è già stato configurato, il server presenta all'utente un'interfaccia per l'amministrazione delle preferenze. Questa riporta i dettagli dei dati precedentemente immessi, con particolare riguardo per le reti salvate. Nel caso una di esse non fosse più accessibile o

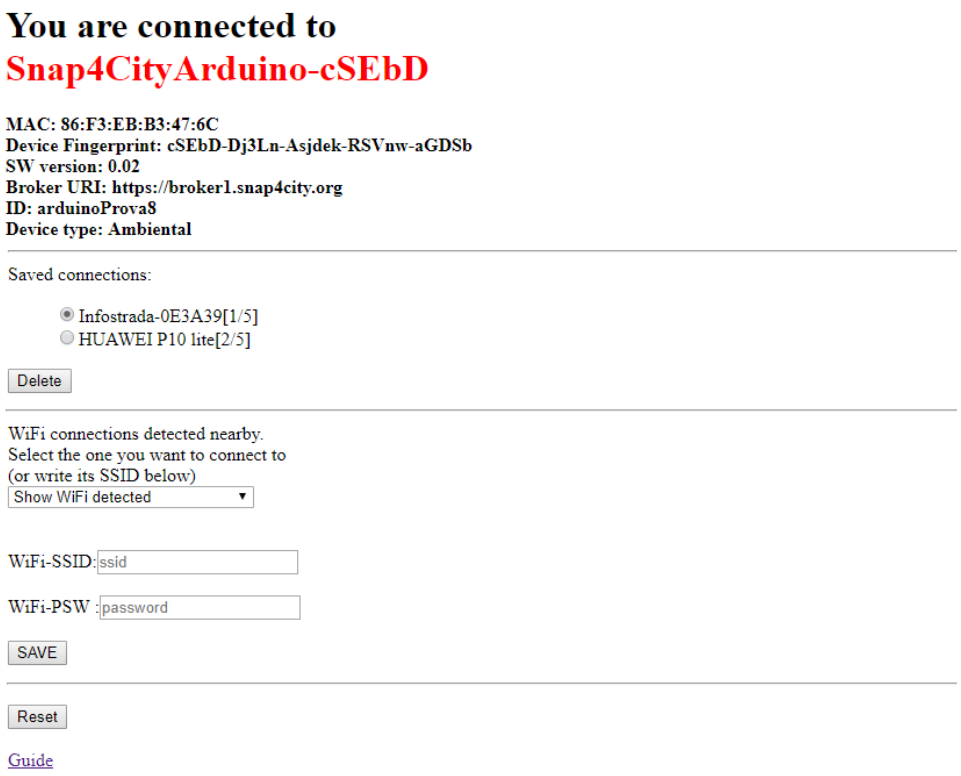


Figura 2.5: Interfaccia di gestione connessioni.

necessaria, l'utente avrà la facoltà di rimuoverla selezionandola ed eseguendo un click sul pulsante `Delete`. Internamente, ne conseguirà una riorganizzazione degli slot dedicati in memoria, supervisionata dalla funzione `handleDelete()`. Inoltre, vengono elencate le nuove reti individuate nelle vicinanze, permettendo la loro aggiunta e di liberare il dispositivo dai vincoli imposti dal raggio di azione di quelle memorizzate. La funzione `handleSaveNewConnection()` provvederà ad allocare adeguatamente la nuova coppia di credenziali. Infine, per consentire la variazione dei restanti parametri e/o del metodo di autenticazione, sarà possibile eseguire un reset del dispositivo. In accordo con le operazioni di scrittura in memoria, la funzione `handleReset()` consisterà nell'inserimento del valore 255 in ogni sua cella.

2.4.3 EEPROM

Per consentire al client di salvare permanentemente i parametri di configurazione immessi dall'utente, si è fatto affidamento sull'utilizzo della libreria `EEPROM` (Electrically Erasable Programmable Read-Only Memory). Questa permette di simulare un disco rigido di dimensioni ridotte, i cui valori sono conservati anche qualora il sistema venga spento [17].

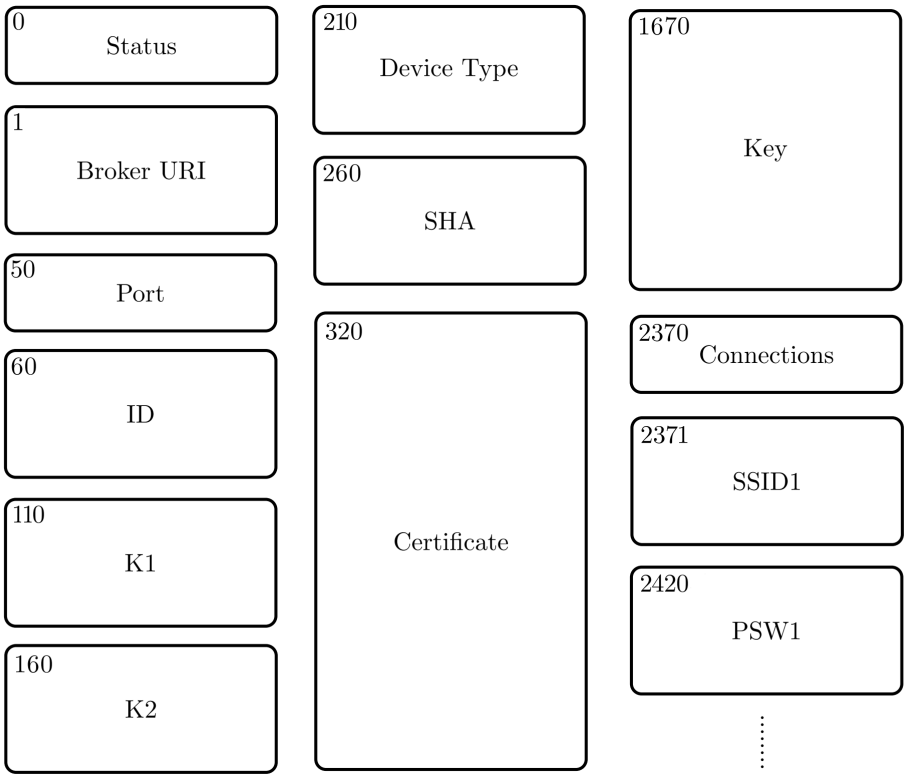


Figura 2.6: Suddivisione della `EEPROM` con valori dei puntatori globali.

Dal momento che il chip offre una quantità di memoria dinamica estremamente limitata, si è dovuto trovare un compromesso tra la sua disponibilità e l'informazione da memorizzare. È stato verificato, infatti, che il volume di allocazioni di memoria riservato all'`EEPROM` viene scalato direttamente dalla

RAM, rendendosi di fatto indisponibile per supportare le successive operazioni. Pertanto, nonostante la libreria ammetta un'espansione massima di 4096 celle, ne sono state adoperate unicamente 2900. Di queste, circa 1300 e 650 sono state destinate, rispettivamente, a certificato e chiave. Le restanti, invece, sono state distribuite per conservare lo stato corrente del dispositivo, il suo ID, il Broker URI con relativa porta, la coppia K1, K2 e fino a 5 configurazioni di rete. Una volta organizzata logicamente la suddivisione interna, si è proceduto a implementare le operazioni di lettura e scrittura. Si è inizialmente supposto che una cella fosse vuota qualora al suo interno contenesse 255, massimo valore rappresentabile con gli 8 bit a disposizione. Grazie a questa assunzione, ogni allocazione associata a un numero differente è stata considerata come portatrice di un'unità informativa. Pertanto, una volta riportati in variabili globali i puntatori alle celle iniziali di ogni campo memorizzato, recuperare un parametro è stato analogo a ricercare, a partire dal relativo puntatore, tutti i dati fino ad una cella pari a 255, contrassegnante la loro fine. Il salvataggio, al contrario, si è rivelato più agevole, poiché è stato sufficiente limitarsi a scrivere un valore per cella a partire dal corrispettivo puntatore. Solo in caso di sovrascrittura si è dovuto prendere un'ulteriore accortezza, ovvero immettere 255 nelle eventuali celle di differenza con il dato precedente.

2.4.4 WiFiClientSecure

Come già preannunciato nel paragrafo 1.3, per poter realizzare un protocollo di comunicazione sicuro, avvalorato dalla mutua autenticazione, si è fatto riferimento alla libreria `WiFiClientSecure`, basata su `axTLS` [18]. Questa fornisce le necessarie funzionalità per mettere in atto i principi costitutivi del protocollo TLS, conseguiti attraverso le seguenti operazioni:

1. Dopo aver istanziato l'oggetto `client`, sono invocati su di esso i metodi preposti a specificarne il certificato e la chiave in uso. Dal momento che l'utente, in fase di configurazione, provvede a caricarli in formato PEM, per non incorrere in errori di compilazione è necessario convertirli in formato DER. Come spiegato nel paragrafo 1.4, questo è possibile eseguendo una decodifica base64 delle credenziali originali.

2. Mediante la funzione `connect()` il server viene contattato in corrispondenza di una porta sicura, nello scenario in esame la 8080, e viene instaurata una connessione.

3. Una volta ottenuto un riscontro positivo, la libreria procede nel completare la TLS *handshake* servendosi dei parametri indicati.

4. Per appurare l'identità del server, tramite la funzione `verify()`, viene eseguito un confronto tra la sua *signature* e il valore dello SHA immesso dall'utente in fase di configurazione.

5. Infine, facendo uso della funzione `write()`, sono inoltrati la richiesta e il contenuto informativo secondo una chiamata `POST`.

Qualora l'utente non fosse in possesso di una certificazione, sarà comunque in grado di dialogare con il server specificando K1 e K2. In questa eventualità la richiesta risultante sarà lievemente differente, includendo al suo interno la definizione dei due parametri. In ogni caso, la struttura della `POST` effettuata sarà analoga a quella descritta nel paragrafo 1.2.

2.4.5 Principali Complicazioni

Come già messo in risalto in precedenza, la maggiore complicazione si è rivelata essere la limitata quantità di memoria fruibile. Nel corso dello

sviluppo, la sua occupazione è stata costantemente monitorata, in quanto causa primaria di gran parte degli esperimenti falliti. Al termine della compilazione, infatti, il suo utilizzo si è sempre aggirato sui 44 KiB, pari al 53% del totale. A tal proposito, nel paragrafo 2.4.3, abbiamo già trattato come a incidere su questo valore sia anche la `EEPROM`. Alla luce di queste statistiche, la problematicità del suo dimensionamento appare ancor più manifesta. Inoltre, un non trascurabile impatto è riconducibile al largo utilizzo del tipo `String`, reso necessario dalla codifica `HTML` e `Javascript` delle pagine del server. I difetti di questa classe sono imputabili all'allocazione dinamica del dato sull'*heap*, sempre meno ottimizzata ad ogni concatenamento con un segmento testuale successivo. All'occorrere di questo evento, lo spazio riservato alla stringa risultante sarà maggiore alla somma delle due iniziali, causando una notevole frammentazione. Una soluzione naïve, seguita in principio, è stata quella di limitarne l'utilizzo, senza poter tuttavia scendere sotto una certa soglia. Difatti, per gestire un'interfaccia utente tanto pragmatica quanto scarna, è comunque richiesto un loro massivo utilizzo, comportante consumi globali fino a 5 MiB. Per scongiurare di incorrere in errori, sono state seguite 3 accortezze:

1. fare uso della `PROGMEM`. Questo modificatore ha consentito di trasferire l'allocazione delle stringhe su cui è stato applicato dalla RAM al segmento riservato alle istruzioni. Sebbene ne sia comportato un considerevole risparmio di memoria, durante l'invio dell'informazione al webserver i tempi di caricamento ne hanno leggermente sofferto, dovendo questo avvenire su due fasi.

2. compilare selezionando l'impostazione `NDEBUG`. Ciò permette di disabilitare le ASSERT macro, utili in fase di sviluppo in quanto fonte di maggiore verbosità durante il debug, risparmiando la memoria a loro

riservata. Numericamente, questo ha consentito di risparmiare ulteriori 2 KiB di RAM.

3. chiudere definitivamente la modalità AP. La coesistenza della modalità Access Point e Station è stata appurata essere impraticabile in fase di stabilimento della connessione. Si è dovuto, pertanto, provvedere esplicitamente a compiere questa transizione a ogni trasmissione del dato.

Una seconda complicazione ha riguardato il corretto svolgimento della procedura di TLS *handshake*, la quale non veniva portata a termine a causa di un fallimento. Dopo alcuni accertamenti, è emerso che la negoziazione dei cifrari non si concludeva positivamente, evidenziando la necessità di abilitare lato server l'utilizzo di *cipher suite* che prevedessero codifica AES, autenticazione mediante SHA1 e scambio della chiave secondo RSA. Queste, inoltre, dovevano essere compatibili con quelle supportate dal chip `ESP8266`.

Capitolo 3

IoT Button ed ESP32

3.1 Specifiche Hardware

A due anni dal lancio del chip `ESP8266`, oggetto della trattazione del capitolo precedente, nel 2016 l'azienda Espressif ha annunciato e reso disponibile sul mercato il suo successore: l'`ESP32`. Tale microcontrollore integra il processore Tensilica Xtensa LX6, sia nella variante single che dual core, capace di operare a 240 MHz. Le complicazioni riconducibili alla scarsa disponibilità di memoria sono state superate, in virtù di un'espansione della SRAM fino a raggiungere i 520 KiB. A differenza del predecessore, introduce un'accelerazione hardware rivolta a supportare gli algoritmi di sicurezza RSA, AES e SHA, nota particolarmente positiva per le finalità descritte in questo elaborato. La connettività, invece, è rimasta immutata, continuando a offrire pieno supporto alle tre varianti `b/g/n` dello standard `IEEE 802.11` [19]. Tale chip è stato in seguito installato su un dispositivo di dimensioni ridotte, lo `Snap4City IoT Button`, pensato per poter essere impugnato senza difficoltà e consentire un suo utilizzo in mobilità.

Figura 3.1: Snap4City IoT Button.

Esso presenta due pulsanti, attraverso i quali l'utente potrà manifestare i propri comandi, e un led a 256 colori, per notificarne l'esito. Le componenti elettroniche, a loro volta, sono state inserite in un comodo contenitore, realizzato mediante una stampa 3D, sul cui retro è stato inciso l'univoco numero seriale.

3.2 Scenari di interesse

Esaminiamo, facendo riferimento a Figura 3.2, il comportamento del dispositivo in risposta a diversi scenari di interesse.

1. Al momento del primo avvio il sistema subentra nello stato di *Dispositivo Resettato*, in cui permane fino a quando non viene rilevata una qualsiasi pressione di un pulsante. A seguito di questa, per quattro minuti, sarà attivato ed esposto l'Access Point. Qualsiasi azione dell'utente, fatta eccezione per quella di *reset*, una volta gestita, condurrà ad uno stato di inattività, detto di *DeepSleep*.

2. Ultimata la configurazione del dispositivo, all'occorrere di una doppia pressione prolungata, sarà possibile accedere nuovamente all'Ac-

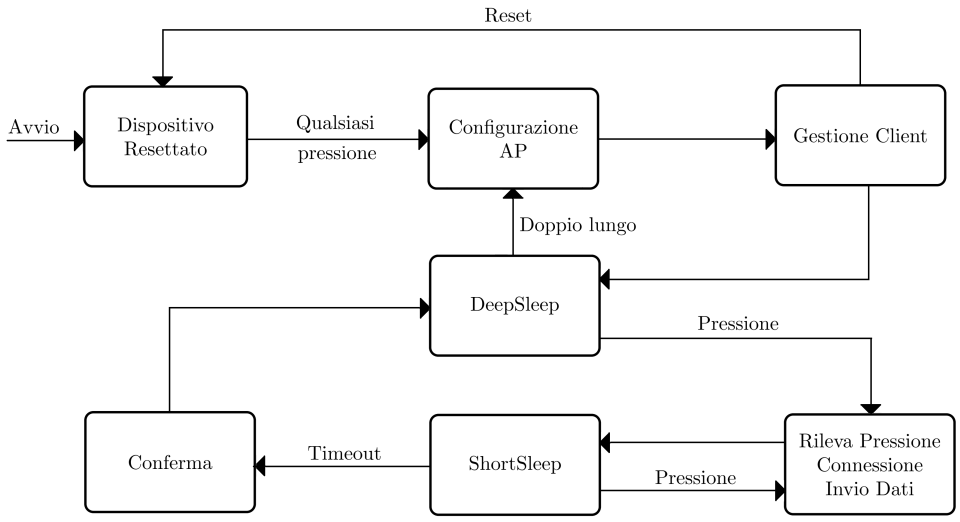


Figura 3.2: Modello secondo la macchina a stati.

cess Point per apportare modifiche alle configurazioni WiFi preesistenti oppure immetterne di nuove.

3. Qualora, invece, sia individuata una pressione di tipologia differente, questa verrà interpretata per permettere di inoltrare il corretto messaggio associato. Ciò avverrà solo in un secondo momento, a seguito dell'instaurazione della connessione con una rete memorizzata.

4. Al termine dell'invio, il dispositivo subentrerà in uno stato di *DeepSleep* ridotto, detto *ShortSleep*, dalla durata massima pari a tre secondi. Se in questo arco di tempo verrà rilevata una nuova interazione dell'utente, si farà ritorno allo stato precedente in modo ciclico. Allo scattare del timeout, d'altro canto, si procederà a trasmettere la conferma attestante la conclusione dello scambio informativo.

3.3 Programmazione ESP32

3.3.1 Pulsanti

Come delineato descrivendo le caratteristiche hardware, l'`IoT Button` espone due pulsanti con i quali l'utente può interagire. Per consentire il maggior numero di combinazioni per ciascuno di essi e distinguere i molteplici scenari, sono state calibrate tre tipologie di pressione: corta, doppia e prolungata. Il computo delle possibili azioni si è, in seguito, accresciuto, contemplando anche la possibilità di premere contemporaneamente i due pulsanti più o meno a lungo. Per permettere un accurato riconoscimento dell'input, non si è ritenuto affidabile ricorrere a una sua misurazione temporale, a causa del fenomeno del *debouncing*. Infatti, benché risultino praticamente impercettibili ai sensi umani, i rimbalzi meccanici degli interruttori possono provocare transizioni spurie. Di conseguenza, si è optato per un approccio incentrato sul campionamento di 6 segnali in ingresso, rilevati ogni 150 ms. Osserviamo più nel dettaglio l'evoluzione delle variabili di stato e i criteri distintivi:

- qualora vengano acquisiti almeno 4 campioni *alti*, si potrà catalogare l'ingresso come una pressione prolungata. Infatti, per ottenere un simile risultato, il pulsante dovrà essere stato premuto per almeno 750 ms.

- qualora venga registrato almeno un campione *alto*, inizierà ad essere monitorato anche il numero di quelli *bassi*. Se a questi non dovesse seguire alcun valore differente, si potrà propendere per un click singolo.

- qualora dopo l'occorrenza di un campione *basso*, come citato nel punto precedente, fosse individuato un ulteriore campione *alto*, ciò indicherebbe una nuova pressione avvenuta, suggerendo un doppio click.

Questa logica, incapsulata nella funzione `readButtonPress()`, viene invocata ogniqualvolta il dispositivo rilevi l'inizio di una pressione da parte dell'utente.

3.3.2 WiFi

Sebbene sia stato necessario risolvere alcuni problemi di disambiguazione all'interno dell'ambiente di sviluppo, la libreria `WiFi` si presenta pressochè analoga alla controparte per `ESP8266`. Ciò ha permesso di preservare le medesime funzioni riguardanti sia la gestione delle reti memorizzate, finalizzata a garantire la connessione a quella più intensa, che quella del dualismo tra le modalità AP e STA. A tal riguardo, le criticità correlate al passaggio dall'una all'altra sembrano essere state sanate, sebbene la documentazione riporti indicazioni controverse. Infatti, benché in essa sia specificato che per eseguire la scansione delle connessioni nei dintorni il dispositivo debba trovarsi in modalità condivisa [20], è invece sufficiente che sia abilitato il suo profilo *server*. Sulla base di questa evidenza sperimentale, è stato possibile disabilitare la modalità STA andando ad abbattere l'impatto energetico, tema largamente discusso nel paragrafo 3.3.6. Ciò è stato agevolato anche dalla strutturazione del codice secondo una macchina a stati, tra i quali quello di *deepsleep* ha maggiormente condizionato la connettività. Infatti, a differenza dell'`ESP8266` dove il dialogo con il router non subiva mai interruzioni, quando il dispositivo entra in questo stato deve necessariamente cessare il collegamento, a causa della disabilitazione del modulo WiFi. Di conseguenza, ad ogni suo risveglio da tale condizione, si dovrà provvedere a effettuare una nuova connessione prima di procedere all'inoltro del messaggio.

3.3.3 WebServer

Nonostante la libreria `ESP8266WebServer` fosse rivolta all'hardware di cui porta il nome, l'implementazione della versione per `ESP32`, nota come `WebServer`, ne ha preservato le peculiarità, rendendo intuitiva la transizione. Analogamente a quanto descritto nel paragrafo 2.4.2, si è proceduto a istanziare un oggetto `server`, a cui sono state associate le pagine che lo costituiscono e le relative funzioni. Per garantire un'esperienza utente quanto più coerente e continuativa nell'interazione con diversi client, è stata riproposta la medesima UI esposta da `Arduino`. L'unica discrepanza consiste nel rilassamento delle condizioni imposte sull'occupazione di memoria, scenario che ha permesso di evitare di avvalersi della `PROGMEM` e di specifiche configurazioni di compilazione.

3.3.4 EEPROM

Dal momento che la libreria `EEPROM`, con cui era stata progettata e suddivisa la memoria in precedenza, non è propria dell'ecosistema `ESP8266`, la sua portabilità non ha comportato complicazioni degne di nota. Sia la logica di lettura che quella di scrittura non ha subito alterazioni, permettendo il riutilizzo di tutte le funzioni già implementate. Alcune variazioni apportate, non imputabili alla libreria stessa, sono, tuttavia, derivate dalle numerose novità introdotte nel modulo `WiFiClientSecure`. Come avremo modo di approfondire nel paragrafo 3.3.5, tale classe propone modifiche radicali, tra cui la necessità di fornire il certificato e la chiave in formato PEM. Inevitabilmente, ciò ha implicato una diversa ripartizione dei settori della memoria, essendo il supporto testuale più dispendioso in termini di lunghezza. Una stringa codificata, infatti, esige un incremento del 33% delle allocazio-

ni richieste dalla corrispettiva rappresentazione in byte. Per poter salvare correttamente questi parametri, pertanto, si è reso necessario aumentare la dimensione dell'`EEPROM`, raggiungendo le 3900 celle. Tuttavia, dal momento che la RAM messa a disposizione dall'`ESP32` è considerevolmente maggiore di quella offerta dall'`ESP8266`, a tale impiego aggiuntivo di spazio in memoria non è conseguito alcun inconveniente.

3.3.5 WiFiClientSecure

La libreria `WiFiClientSecure`, sebbene contraddistinta dal medesimo nome per entrambi i chip, si è rivelata essere la causa delle maggiori rettifiche operate durante il *porting* del codice verso il nuovo client. Queste sono imputabili a una radicale ristrutturazione del suo nucleo operativo, non più basato su `AxTLS`. Con l'intento di adeguare la libreria a più recenti standard di sicurezza, infatti, gli sviluppatori hanno ritenuto appropriato adottare `MbedTLS` [21]. Ne è conseguito che, ad esempio, avendo Google dimostrato potenziali vulnerabilità nell'utilizzo dello SHA1 [22], la funzione `verify()` sia stata deprecata e, al momento, non ne venga fornita alcuna implementazione correttamente funzionante secondo le nuove direttive. Per adempiere allo stesso compito viene, invece, richiesto di fornire la *certificate authority* nella sua interezza. Inoltre, come già accennato in merito alla `EEPROM`, il formato della coppia certificato e chiave è passato da DER a PEM, rendendo inevitabile la ristrutturazione del codice. Alla funzione `connect()`, rimasta pressoché invariata, è stato invece associato un timeout pari a 1 secondo, in cui portare a termine l'operazione. Per rendere più stabile la comunicazione con il server, si è ritenuto opportuno eseguire fino a tre tentativi di inoltro, prima di notificarne il fallimento. Ciò ha permesso di irrobustire considerevolmente l'affidabilità della trasmissione, spesso complicata da reti poco

stabili e operazioni molto dispendiose in termini sia di complessità che di tempo. Una novità rispetto alla logica dell'`ESP8266` è ravvisabile nell'invio di un messaggio di convalida una volta trascorsi 3 secondi dall'ultima attività dell'utente, senza che questi ne sia esplicitamente informato. Ciò ha richiesto, pertanto, di manipolare ulteriormente i metodi precedentemente definiti in modo da impedire che nessun led fosse coinvolto in tale procedura. La necessità di avere simile riscontro deriva dalle applicazioni associate, le quali esigono che sia fornita una terminazione esplicita del segnale precedente per riuscire a simulare l'andamento dell'impulso.

3.3.6 Principali complicazioni

In questo inedito scenario, le caratteristiche del nuovo hardware preso in esame hanno decretato come prioritaria la ricerca di una soluzione a basso impatto energetico. A differenza di `Arduino`, infatti, il dispositivo non è stato progettato per essere alimentato in modo continuativo; al contrario, dovrebbe cercare di ottimizzare il consumo delle batterie, garantendo un loro utilizzo il più prolungato possibile nel tempo. Per conseguire questi obiettivi, si è rivelato necessario adottare le seguenti strategie, mirate a minimizzare l'impiego delle risorse a disposizione.

1. Provvedere a formalizzare un modello ispirato alla macchina a stati, le cui transizioni hanno luogo nella funzione `loop`. In tal modo si è potuto evitare di confidare in implementazioni, meno prestazionali, di cicli aggiuntivi. Dovendo poi il dispositivo trascorrere gran parte del suo arco vitale in uno stato di inattività, è stato essenziale garantire che in quegli intervalli il consumo fosse ridotto al minimo indispensabile. Ciò è stato realizzato avvalendosi della modalità di *deepsleep*, durante la quale tutti i moduli e i *core* della CPU sono disattivati, la memoria

volatile viene persa ed entra in vigore un co-processore *ultra-low-power*. Questa filosofia è stata riproposta, in scala minore, anche nel periodo di tre secondi presente tra un inoltro e la conseguente conferma, in cui può avvenire un risveglio imputabile a un input esterno o allo scadere del timer.

2. Regolare le tonalità illuminate dal led, i cui valori sono stati progressivamente smorzati. Ciò ha comportato numerose rimodulazioni dei tre canali cromatici, fino ad adottare gradazioni che prevedessero una loro somma non maggiore di 32. Nonostante questa limitazione, sono stati comunque individuati 9 colori sufficientemente caratteristici.

3. Limitare la permanenza nello stato di configurazione. Benché si supponga che l'utente vi faccia ricorso in modo assai sporadico, deve essere comunque garantito un adeguato intervallo di funzionamento per compilare correttamente i parametri. Alla luce dei notevoli consumi in questa fase, tuttavia, per scongiurare che la sua esposizione si protragga eccessivamente, è stato inizialmente introdotto un timeout automatico allo scadere di quattro minuti, a cui è seguita la possibilità di annullare l'operazione dall'interfaccia grafica.

4. In merito al punto precedente, avendo delineato come caso peggiore lo stazionamento ininterrotto in modalità AP per 240 secondi, in tale arco di tempo il led è stato istruito a lampeggiare ogni secondo, dimezzandone il fabbisogno energetico.

5. In accordo con quanto descritto nel paragrafo 3.3.2, si è inoltre assicurato che le modalità STA e AP non fossero attive contemporaneamente.

6. La potenza trasmissiva di quest'ultima, infine, è stata limitata a un raggio molto ristretto, presupponendo che l'utente vi effettui l'accesso tramite un dispositivo nelle sue vicinanze. Per garantire che ciò non avesse ripercussioni negative sul risultato della scansione delle reti circostanti, in tale frangente la potenza trasmissiva è stata nuovamente regolata al massimo.

Inoltre, dal momento che la funzione `verify()` non offre un'implementazione adeguata, il controllo aggiuntivo sulla validità del certificato lato server è stato sospeso. Difatti, nonostante sia possibile specificare una *certificate authority*, l'accertamento restituirà sempre un risultato negativo per certificati *self-signed*, come nel caso delle credenziali del Broker.

Capitolo 4

Conclusioni

Lo studio e l'implementazione discussi in questo elaborato non hanno consentito unicamente di accrescere la sicurezza dello scambio informativo. La soluzione proposta, infatti, prevedendo un paradigma di mutua autenticazione basato sul protocollo HTTPS, ha permesso, inoltre, di adeguarsi alla normative indicate nel nuovo Regolamento Generale sulla Protezione dei Dati, più comunemente noto secondo l'acronimo GDPR. Questo, dalla sua entrata in vigore il 25 Maggio 2018, sancisce proprio, tra le molteplici clausole, che le aziende debbano attuare un processo di cifratura dei dati, limitando solo ai diretti interessati la loro lettura.

Il primo capitolo riassume, in modo conciso ma puntuale, le nozioni propedeutiche per svolgere un efficace studio preliminare della tematica in esame. Ciò ha permesso, in un primo momento, di comprendere la politica dell'`Orion Broker` e, in seguito, di operare una scelta ponderata nella generazione di chiavi e certificati.

Il secondo capitolo si incentra sull'attuazione del paradigma di mutua autenticazione per mezzo di `Arduino Uno` e del chip `ESP8266`. Tale lavoro è confluito, infine, nella realizzazione di una dashboard, atta a tracciare l'andamento temporale dei parametri rilevati dal sensore. Questa, integrata nella piattaforma `Snap4City`, permette, a tutti gli effetti, di monitorare uno scenario concreto e completamente operativo.

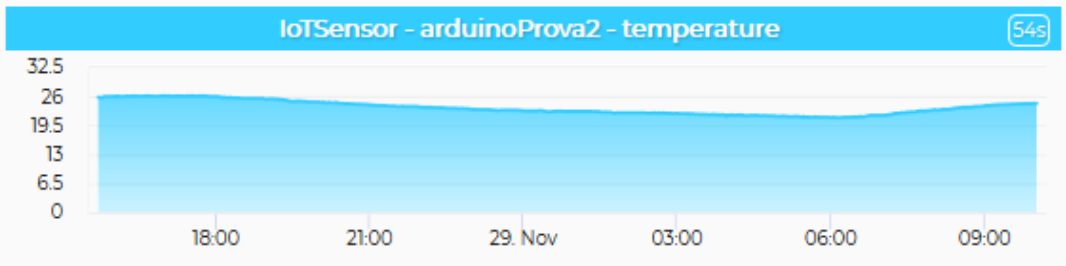


Figura 4.1: Dettaglio dell'andamento della temperatura.

Il terzo capitolo si focalizza, invece, sui pregi e difetti della nuova soluzione hardware adottata: il chip `ESP32`. Il suo sviluppo software si è concluso con la configurazione di un prototipo, impiegabile nell'ecosistema della piattaforma cloud, finalizzato alla trasmissione di diverse tipologie di segnale. Come *proof of concept*, infatti, è stata riservata una dashboard, capace di rappresentare gli impulsi inoltrati durante la giornata.

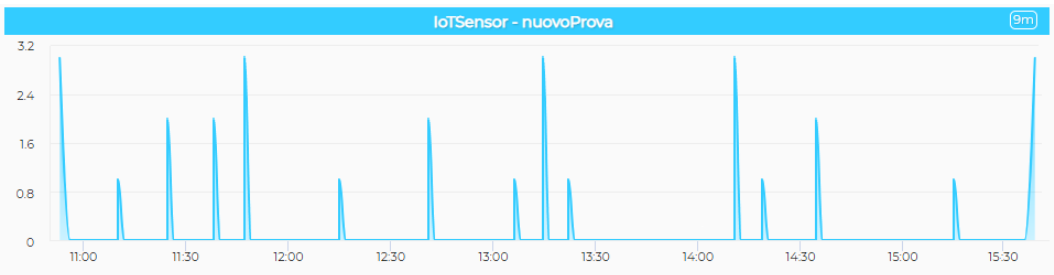


Figura 4.2: Dettaglio degli impulsi rilevati nel tempo.

Sebbene quanto conseguito sia sicuramente degno di nota, ciò non deve rappresentare un traguardo, bensì un punto di partenza. L'implementazione proposta, infatti, non solo risolve la problematica iniziale, ma si lascia aperta a future ottimizzazioni. Il codice sorgente, pertanto, è stato pubblicato in forma open-source presso i canali ufficiali del DISIT Lab [23] [24], dove è inoltre possibile reperire la guida tecnica e la guida per l'utente finale.

Bibliografia

[1] Eclipse Foundation. *Key Trends from the IoT Developer Survey 2018.* https://blogs.eclipse.org/post/benjamin-cab%C3%A9/key-trends-iot-developer-survey-2018.

[2] Internet of Things Agenda. *Definition of Internet of Things.* https://internetofthingsagenda.techtarget.com/definition/Internet-of-Things-IoT.

[3] Gartner. *Gartner Says 8.4 Billion Connected "Things" Will Be in Use in 2017, Up 31 Percent From 2016.* https://gtnr.it/2Mcqz56.

[4] Gartner. *Leading the IoT.* https://www.gartner.com/imagesrv/books/iot/iotEbook_digital.pdf.

[5] Microsoft Visual Studio. *Scott Hanselman's best demo. IoT, Azure, Machine Learning and more.* https://www.youtube.com/watch?v=u5oTz1e5qqE.

[6] Oleg Šelajev. *The S in the IoT stands for Security.* https://twitter.com/shelajev/status/796685986365325312?lang=en.

[7] Symatec. *Internet Security Threat Report Volume 23.* http://images.mktgassets.symantec.com/Web/Symantec/

%7B3a70beb8-c55d-4516-98ed-1d0818a42661%7D_ISTR23_Main-FINAL-APR10.pdf?aid=elq.

[8] Kaspersky. *New IoT-malware grew three-fold in H1 2018.* https://www.kaspersky.com/about/press-releases/2018_new-iot-malware-grew-three-fold-in-h1-2018.

[9] DISIT Lab. *Snap4City - scalable Smart aNalytic APplication builder for sentient Cities.* https://www.snap4city.org/drupal/node/1.

[10] Fiware. *NGSI Context Management Information Model.* http://aeronbroker.github.io/Aeron/.

[11] Ivan Ristić. *Bulletproof SSL and TLS.* Feisty Duck, 2015.

[12] Espressif. *WiFiClientSecure ESP32.* https://github.com/espressif/arduino-esp32/tree/master/libraries/WiFiClientSecure.

[13] ESP8266 Community Forum. *WiFiClientSecure ESP8266.* https://github.com/esp8266/Arduino/blob/master/libraries/ESP8266WiFi/src/WiFiClientSecure.h/.

[14] CertSimple. *So you're making an RSA key for an HTTPS certificate. What key size do you use?* https://certsimple.com/blog/measuring-ssl-rsa-keys.

[15] Digicert. *The Math Behind Estimations to Break a 2048-bit Certificate.* https://www.digicert.com/TimeTravel/math.htm.

[16] Arduino. *Scheda tecnica Arduino Uno.* https://store.arduino.cc/arduino-uno-rev3.

[17] Arduino. *EEPROM Library.* https://www.arduino.cc/en/Reference/EEPROM.

[18] axTLS. *axTLS Embedded SSL.* http://axtls.sourceforge.net/.

[19] Espressif. *ESP32 Datasheet.* https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf.

[20] Espressif. *ESP-IDF Programming Guide.* https://dl.espressif.com/doc/esp-idf/latest/api-guides/wifi.html.

[21] mbedTLS. *mbedTLS Homepage.* https://tls.mbed.org/.

[22] Google. *Announcing the first SHA1 collision.* https://security.googleblog.com/2017/02/announcing-first-sha1-collision.html.

[23] DISIT. *Snap4City: Arduino and ESP8266 IOT Device NGSI.* https://www.snap4city.org/drupal/node/329.

[24] DISIT. *Snap4All IOT Button: based on ESP32, NGSI compliant secure connection.* https://www.snap4city.org/drupal/node/276.